Malleable Proof Systems and Applications

Full version of an extended abstract published in Proceedings of Eurocrypt 2012, Springer-Verlag, 2012. Available from the IACR Cryptology ePrint Archive as Report 2012/012.

Malleable Proof Systems and Applications

Melissa Chase
Microsoft Research Redmond
melissac@microsoft.com

Markulf Kohlweiss
Microsoft Research Cambridge
markulf@microsoft.com

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

Sarah Meiklejohn*
UC San Diego
smeiklej@cs.ucsd.edu

January 16, 2012

Abstract

Malleability for cryptography is not necessarily an opportunity for attack, but in many cases a potentially useful feature that can be exploited. In this work, we examine notions of malleability for non-interactive zero-knowledge (NIZK) proofs. We start by defining a malleable proof system, and then consider ways to meaningfully *control* the malleability of the proof system, as in many settings we would like to guarantee that only certain types of transformations can be performed. We also define notions for the cases in which we do not necessarily want a user to know that a proof has been obtained by applying a particular transformation; these are analogous to function/circuit privacy for encryption.

As our motivating application, we consider a shorter proof for verifiable shuffles. Our controlled-malleable proofs allow us for the first time to use one compact proof to prove the correctness of an entire multi-step shuffle. Each authority takes as input a set of encrypted votes and a controlled-malleable NIZK proof that these are a shuffle of the original encrypted votes submitted by the voters; it then permutes and re-randomizes these votes and updates the proof by exploiting its controlled malleability. As another application, we generically use controlled-malleable proofs to realize a strong notion of encryption security.

Finally, we examine malleability in existing proof systems and observe that Groth-Sahai proofs are malleable. We then go beyond this observation by characterizing all the ways in which they are malleable, and use them to efficiently instantiate our generic constructions from above; this means we can instantiate our proofs and all their applications using only the Decision Linear (DLIN) assumption.

*Work done as an intern at Microsoft Research Redmond

Contents

1 Introduction

Let L be a language in NP. For concreteness, consider the language of Diffie-Hellman tuples: $(G, g, X, Y, Z) \in L_{DH}$ if there exist (x, y) such that g, X, Y, Z are elements of the group G, $X = g^x$, $Y = g^y$, and $Z = g^{xy}$. Suppose that we have a polynomial time prover P, and a verifier V, and P wants to convince V that $(G, g, X, Y, Z) \in L_{DH}$. Does the efficient prover need to know the values (x, y) in order to convince the verifier? Not necessarily. Suppose that P is in possession of a non-interactive zero-knowledge (NIZK) proof π' that another tuple, $(G, g, X', Y', Z') \in L_{DH}$; suppose in addition that P happens to know (a, b) such that $X = (X')^a$, $Y = (Y')^b$, and $Z = (Z')^{ab}$. Can he, using the fact that he knows (a, b), transform π' into a NIZK π for the related instance (G, g, X, Y, Z)? In the sequel, we say that a proof system is *malleable* if it allows a prover to derive proofs of statements (such as $(G, g, X, Y, Z) \in L_{DH}$) not just from witnesses for their truth, but also from proofs of related statements (such as the proof π' that $(G, g, X', Y', Z') \in L_{DH}$).

In this paper, we consider malleability for non-interactive zero-knowledge proof systems. Our contributions are threefold: (1) definitions; (2) constructions; and (3) applications.

Motivating application. Why is malleability an interesting feature for non-interactive zero-knowledge proof systems? Let us present, as a motivating application, a verifiable vote shuffling scheme that becomes much more efficient if constructed using malleable proofs.

In a vote shuffling scheme, we have a set of encrypted votes $(v_1, \ldots, v_n)$ submitted by n voters; each vote v_i is an encryption of the voter's ballot under some trusted public key pk. The set of encrypted votes is then re-randomized[1] and shuffled, in turn, by several shuffling authorities. More precisely, let $(v_1^{(0)}, \ldots, v_n^{(0)}) = (v_1, \ldots, v_n)$; then each authority A_j takes as input $(v_1^{(j-1)}, \ldots, v_n^{(j-1)})$, picks a random permutation ρ and outputs $(v_1^{(j)}, \ldots, v_n^{(j)}) = (\tilde{v}_{\rho(1)}^{(j)}, \ldots, \tilde{v}_{\rho(n)}^{(j)})$, where $\tilde{v}_i^{(j)}$ is a randomization of $v_i^{(j-1)}$. At the end, the final set of encrypted votes $(v_1^{(\ell)}, \ldots, v_n^{(\ell)})$ is decrypted (for example, by a trustee who knows the decryption key corresponding to pk, or via a threshold decryption protocol) and the election can be tallied.

It is easy to see that, if we are dealing with an honest-but-curious adversary, this scheme guarantees both correctness and privacy as long as at least one of the authorities is honest. To make it withstand an active adversary, however, it is necessary for all participants (both the voters and the shuffling authorities) to prove (using a proof system with appropriate, technically subtle soundness and zero-knowledge guarantees) that they are correctly following the protocol. If these proofs are non-interactive, then the protocol gets the added benefit of being universally verifiable: anyone with access to the original encrypted votes and the output and proofs of each authority can verify that the votes were shuffled correctly. Thus, any party wishing to verify an election with n voters and ℓ shuffling authorities (and ℓ can potentially be quite large, for example a large polynomial in n for cases where a small group is voting on a very sensitive issue) will have to access $\Omega(n\ell)$ data just to read all the proofs.

Can the proof that the verifier needs to read be shorter than that? The statement that needs to be verified is that the ciphertexts $(v_1^{(\ell)}, \ldots, v_n^{(\ell)})$ can be obtained by randomizing and permuting the original votes $(v_1, \ldots, v_n)$. The witness for this statement is just some permutation (that is obtained by composing the permutations applied by individual authorities) and randomness that went into randomizing each ciphertext (that can be obtained by applying the group operation repeatedly to the randomness used by each authority); thus, ignoring the security parameter, the

[1] It is therefore important that the encryption scheme used is randomizable, so that on input a ciphertext $c = \mathsf{Enc}_{pk}(m; r)$ and randomness r' one can compute $c' = \mathsf{Enc}_{pk}(m; r * r')$, where $*$ is some group operation.

length of the witness can potentially be only $O(n)$.[2]

Of course, no individual authority knows this witness. But each authority A_j is given a proof π_{j-1} that, up until now, everything was permuted and randomized correctly. Using controlled malleable proofs, from this π_{j-1} and its own secret permutation ρ_j and vector of random values $(r_1^{(j)}, \ldots, r_n^{(j)})$, A_j should be able to compute the proof π that his output is a permutation and randomization of the original votes.

In this paper, we give a construction that roughly corresponds to this outline, and prove its security. We must stress that even though this construction is a more or less direct consequence of the new notion of controllable malleability, and therefore may seem obvious in hindsight, it is actually a significant breakthrough as far as the literature on efficient shuffles is concerned: for the first time, we obtain a non-interactive construction in which the complexity of verifying the tally with ℓ authorities is not ℓ times the complexity of verifying the tally with one authority!

Our definitions. Care needs to be taken when defining malleable NIZKs suitable for the above application. We first need malleability itself: from an instance x' and a proof π' that $x' \in L$, we want to have an efficient algorithm ZKEval that computes another instance $x = T(x')$ and a proof π that $x \in L$, where T is some transformation (in the above example, x' may be a set of ciphertexts, and T is a re-randomization and permutation of these ciphertexts). We want the resulting proof to be *derivation private*, so that, from x and π, it is impossible to tell from which T and x' they were derived. (In the above example, it should be impossible to tell how the ciphertexts were shuffled.) Finally, we want to ensure that the proof system is sound, even in the presence of a zero-knowledge simulator that provides proofs of adversarially chosen statements (so that we can relate the real-world experiment where the adversary participates in shuffling the ciphertexts to an ideal-world process that only has access to the final tally). To this end, we define *controlled malleability* (as opposed to malleability that is out of control!) that guarantees that, from proofs computed by an adversary, an extractor (with a special extracting trapdoor) can compute either a witness to the truth of the statement, or the transformation T and some statement for which the simulator had earlier provided a proof.

Our definitional approach to derivation privacy is inspired by circuit privacy for fully homomorphic encryption [27, 41, 40, 13], also called function privacy or unlinkability. Our definitional approach to controlled malleability is inspired by the definitions of HCCA (homomorphic-CCA) secure encryption due to Prabhakaran and Rosulek [37]; it is also related to the recently proposed notion of targeted malleability due to Boneh, Segev, and Waters [12]. (See Appendix Efor more detailed comparison with these notions.)

Our construction. Our construction of controlled-malleable and derivation-private NIZK proof systems consists of two steps. First, in Section 3, we show how to construct a controlled-malleable derivation-private NIZK from any derivation-private non-interactive witness-indistinguishable (NIWI) proof system and secure signature scheme. Then, in Section 4.1 we show how to instantiate the appropriate NIWI proof system and signature scheme using the Groth-Sahai proof system [34] and a recent structure-preserving signature due to Chase and Kohlweiss [15]; this combination means we can instantiate our proofs (and in fact all of the constructions in our paper) using the Decision

[2]Here we use a very simple approach to proving a shuffle in which we represent the permutation as a matrix, thus the length of a single shuffle proof is $O(n^2)$. This could potentially be improved using more sophisticated verifiable shuffle techniques as we will mention later. Additionally, because we want to be able to verify the fact that each authority participated in the shuffle, we will include a public key for each authority involved and the size will actually grow to $O(n^2 + \ell)$.

Linear (DLIN) assumption [11]. The size of the resulting proof is linear in the size of the statement, although the size of the structure-preserving signature does make it admittedly much less efficient than Groth-Sahai proofs alone.

At the heart of our construction is the observation that the Groth-Sahai (GS) proof system is malleable in ways that can be very useful. This feature of GS proofs has been used in prior work in a wide variety of applications: Belenkiy et al. [8] use the fact that the GS proof system can be randomized in order to construct delegatable anonymous credentials; Dodis et al. [20] uses homomorphic properties of GS proofs in order to create a signature scheme resilient to continuous leakage; Acar and Nguyen [7] use malleability to delegate and update non-membership proofs for a cryptographic accumulator in their implementation of a revocation mechanism for delegatable anonymous credentials; and Fuchsbauer [24] uses malleability to transform a proof about the contents of a commitment into a proof of knowledge of a signature on the committed message in his construction of commuting signatures.

Armed with an appropriate construction of a controlled-malleable and derivation-private NIZK, we proceed, in Section 6, to consider the problem of obtaining a verifiable shuffle with compact proofs. We formally define this concept, describe a generic construction from a semantically-secure encryption scheme and a controlled-malleable and derivation-private NIZK following the outline above, and finally argue that we can in fact construct such a proof system for the appropriate set of transformations based on the instantiation described in Section 4.1.

An application to encryption. Can controlled malleability of NIZKs give us controlled malleability for encryption? That is to say, can we achieve a meaningful notion of adaptively secure encryption, even while allowing computations on encrypted data? Similarly to controlled malleability for proofs, we define in Section 5 controlled malleability for encryption (directly inspired by the notion of HCCA security; in this, our work can be considered closely related to that of Prabhakaran and Rosulek), and show a *general* method for realizing it for broad classes of unary transformations, using a semantically secure encryption scheme with appropriate homomorphic properties and a controlled-malleable and derivation-private NIZK for an appropriate language as building blocks. Our construction follows easily from these properties, resulting in a much simpler proof of security than was possible in previous works. (We note that our methods do not extend to n-ary transformations for $n > 1$, because the same limitations that apply for HCCA security, pointed out by Prabhakaran and Rosulek, also apply here. The work of Boneh et al. overcomes this and allows for binary transformations as well, with the sacrifice that, unlike both our scheme and the Prabhakaran-Rosulek scheme, the encryption scheme can no longer satisfy function privacy.)

Related work on shuffling ciphertexts. Shuffles and mixing in general were introduced by Chaum in 1981 [16], and the problem of verifiable shuffles was introduced by Sako and Kilian in 1995 [39]; the work on verifiable shuffles in the ensuing sixteen years has been extensive and varied [2, 26, 6, 35, 30, 25, 42, 32]. In 1998, Abe [1] considered the problem of compact proofs of shuffles. Unlike our non-interactive solution, his solution is based on an interactive protocol[3] wherein all mixing authorities must jointly generate a proof with size independent of ℓ; in comparison, our solution allows authorities to be offline before and after shuffling the ciphertexts. In terms of approaches most similar to our own, Furukawa and Sako [26] use a permutation matrix to shuffle

[3]The protocol could in fact be made non-interactive, but only using the Fiat-Shamir heuristic [23] and thus the random oracle model.

the ciphertexts; they then prove that the matrix used was in fact a permutation matrix, and that it was applied properly. Most recently, Groth and Lu [32] give a verifiable shuffle that is non-interactive (the only one to do so without use of the Fiat-Shamir heuristic [23]), uses pairing-based verifiability, and obtains $O(n)$ proof size for a single shuffle. The advantage, as outlined above, that our construction has over all of these is that one proof suffices to show the security of the entire shuffle; we do not require a separate proof from each mix server. An interesting open problem is to see if there is some way to combine some of these techniques with an appropriate controlled-malleable proof system to obtain a multi-step shuffle with optimal proof size $O(n + \ell)$.

2 Definitions and Notation

Our definitional goal is to formulate what it means to construct a proof of a particular statement using proofs of related statements. Let $R(\cdot, \cdot)$ be some relation that is polynomial-time computable in the size of its first input; in the sequel we call such a relation an *efficient* relation. Associated with R, there is an NP language $L_R = \{x \mid \exists\, w \textit{ such that } R(x, w) = \mathit{TRUE}\}$.[4] For example, let $R(x, w)$ be a relation that holds if the witness $w = (a, b)$ demonstrates that the instance $x = (G, g, A, B, C)$ is a Diffie-Hellman tuple; i.e. it holds if $g, A, B, C \in G$ and $A = g^a$, $B = g^b$, $C = g^{ab}$. Then the language associated with R is L_{DH} defined in the introduction. We often write $(x, w) \in R$ to denote that $R(x, w) = \mathit{TRUE}$.

Let $T = (T_x, T_w)$ be a pair of efficiently computable n-ary functions, $T_x : \{\{0,1\}^*\}^n \to \{0,1\}^*$, $T_w : \{\{0,1\}^*\}^n \to \{0,1\}^*$. In what follows, we refer to such a tuple T as an n-ary *transformation*.

Definition 2.1. *An efficient relation R is* closed *under an n-ary transformation $T = (T_x, T_w)$ if for any n-tuple $\{(x_1, w_1), \ldots, (x_n, w_n)\} \in R^n$, the pair $(T_x(x_1, \ldots, x_n), T_w(w_1, \ldots, w_n)) \in R$. If R is closed under T, then we say that T is* admissible *for R. Let $\mathcal{T}$ be some set of transformations; if for every $T \in \mathcal{T}$, T is admissible for R, then $\mathcal{T}$ is an* allowable *set of transformations.*

For example, for the DH relation R described above, consider $T = (T_x, T_w)$ where for some (a', b'), $T_x(G, g, A, B, C) = (G, g, A^{a'}, B^{b'}, C^{a'b'})$ and $T_w(a, b) = (aa', bb')$; then the Diffie-Hellman relation R is closed under transformation T, and additionally the set $\mathcal{T}$ of transformations of this form (i.e., where there is a transformation T corresponding to any pair (a', b')) is an allowable set of transformations.

Our goal is to define non-interactive zero-knowledge and witness-indistinguishable proof systems for efficient relations R that are (1) malleable with respect to an allowable set of transformations $\mathcal{T}$; that is to say, for any $T \in \mathcal{T}$, given proofs for $x_1, \ldots x_n \in L_R$, they can be transformed into a proof that $T_x(x_1, \ldots, x_n) \in L_R$; and (2) derivation-private; that is to say, the resulting proof cannot be distinguished from one freshly computed by a prover on input $(T_x(x_1, \ldots, x_n), T_w(w_1, \ldots, w_n))$. Before we can proceed, however, we need to recall the definition of a non-interactive zero-knowledge proof system.

A proof system for an efficient relation R allows a prover to prove that a value x is in the associated language L_R. A non-interactive (NI) proof system with efficient provers [10, 21] consists of three PPT algorithms: the algorithm $\mathsf{CRSSetup}(1^k)$ that generates a common reference string (CRS) σ_{crs}, the algorithm $\mathcal{P}(\sigma_{\text{crs}}, x, w)$ that outputs a proof π that $x \in L_R$, and the algorithm $\mathcal{V}(\sigma_{\text{crs}}, x, \pi)$ that verifies the proof; such a proof system must be complete (meaning the verifier will always accept an honestly generated proof) and sound (meaning that a verifier cannot be fooled into accepting a proof for a false statement). A NI zero-knowledge proof (NIZK) [29, 10], additionally requires the existence of a simulator S that can generate proofs without access to a

[4] Without the restriction that R is efficient in its first input, the resulting language won't necessarily be in NP.

witness, while a NI witness-indistinguishable proof system [22] has the requirement that proofs generated using two different witnesses for the same x are indistinguishable from each other. A NI proof of knowledge [29, 9] additionally requires an efficient extractor algorithm E that, on input a proof that $x \in L_R$, finds a witness for the instance x.

Let us recall the formal definitions for non-interactive zero-knowledge proofs of knowledge (NIZKPoK) systems and non-interactive witness-indistinguishable proofs of knowledge (NIWIPoK) systems that we will use throughout. Definition 2.2 below is an amalgamation of definitions found in prior work.We use the original definitions for completeness and soundness of non-interactive proof systems in the common-reference-string model [10]. The version of the definition of zero-knowledge for NIZK we give is originally due to Feige, Lapidot and Shamir (FLS) [21]; they call it "adaptive multi-theorem NIZK." We also use the FLS definition of witness indistinguishability. The version of knowledge extraction we use is a generalization of the definition of knowledge extraction given by Groth, Ostrovsky and Sahai (GOS) [33]: they defined the notion of *perfect* knowledge extraction, while here we find it useful to generalize it, in the straightforward way, to when it is not perfect. We find it convenient for our presentation to put together all these concepts into just one definition.

Definition 2.2 (Non-interactive proof systems)**.** *A set of algorithms* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *constitute a non-interactive (NI) proof system for an efficient relation R with associated language L_R if completeness and soundness below are satisfied. A NI proof system is exractable if, in addition, the extractability property below is satisfied. A NI proof system is witness-indistinguishable (NIWI) if the witness-indistinguishability property below is satisfied. An NI proof system is zero-knowledge (NIZK) if the zero-knowledge property is satisfied. A NIZK proof system that is also extractable constitutes a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system. A NIWI proof system that is also extractable constitutes a non-interactive witness-indistinguishable proof of knowledge (NIWIPoK) system.*

1. *Completeness. For all* $\sigma_{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ *and* $(x, w) \in R$, $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$ *for all proofs* $\pi \xleftarrow{\$} \mathcal{P}(\sigma_{crs}, x, w)$.

2. *Soundness. For all PPT* $\mathcal{A}$*, and for* $\sigma_{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$*, the probability that* $\mathcal{A}(\sigma_{crs})$ *outputs* (x, π) *such that* $x \notin L$ *but* $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$*, is negligible.* Perfect *soundness is achieved when this probability is 0.*

3. *Extractability. There exists a polynomial-time extractor algorithms* $E = (E_1, E_2)$ *such that* $E_1(1^k)$ *outputs* (σ_{ext}, τ_e)*, and* $E_2(\sigma_{ext}, \tau_e, x, \pi)$ *outputs a value w such that (1) a* σ_{ext} *output by* $E_1(1^k)$ *is indistinguishable from* σ_{crs} *output by* $\mathsf{CRSSetup}(1^k)$*; (2) for all PPT* $\mathcal{A}$*, the probability that* $\mathcal{A}(\sigma_{ext}, \tau_e)$ *(where* $(\sigma_{ext}, \tau_e) \xleftarrow{\$} E_1(1^k)$*) outputs* (x, π) *such that* $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$ *and* $R(x, E_2(\sigma_{ext}, \tau_e, x, \pi)) = 0$*, is negligible.* Perfect *extractability is achieved if this probability is 0, and* σ_{ext} *is distributed identically to* σ_{crs}.

4. *Witness indistinguishability. For all* (x, w_1, w_2) *such that* $(x, w_1), (x, w_2) \in R$*, the tuple* (σ_{crs}, π_1) *is indistinguishable from* (σ_{crs}, π_2) *where* $\sigma_{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$*, and for* $i \in \{1, 2\}$*,* $\pi_i \xleftarrow{\$} \mathcal{P}(\sigma_{crs}, x, w_i)$. Perfect *witness indistinguishability is achieved when these two distributions are identical.*

5. *Zero knowledge. There exists a polynomial-time simulator algorithm* $S = (S_1, S_2)$ *such that* $S_1(1^k)$ *outputs* (σ_{sim}, τ_s)*, and* $S_2(\sigma_{sim}, \tau_s, x)$ *outputs a value* π_s *such that for all* $(x, w) \in R$ *and PPT adversaries* $\mathcal{A}$*, the following two interactions are indistinguishable: in the first, we*

compute $\sigma_{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ *and give* $\mathcal{A}$ σ_{crs} *and oracle access to* $\mathcal{P}(\sigma_{crs}, \cdot, \cdot)$ *(where* $\mathcal{P}$ *will output* $\perp$ *on input* (x, w) *such that* $(x, w) \notin R$*); in the second, we compute* (σ_{sim}, τ_s) *and give* $\mathcal{A}$ σ_{sim} *and oracle access to* $S(\sigma_{sim}, \tau_s, \cdot, \cdot)$*, where, on input* (x, w)*,* S *outputs* $S_2(\sigma_{sim}, \tau_s, x)$ *if* $(x, w) \in R$ *and* $\perp$ *otherwise.* Perfect *zero-knowledge is achieved if for all* $(x, w) \in R$*, these interactions are distributed identically.*

Note that in the extractability definition above, we allow the adversary access to the extractability trapdoor. This makes the definition somewhat stronger than seemingly necessary. However, the proof systems we use as a building block in this paper, as well as the ones that we construct, all satisfy this stronger property.

Next, we define a malleable proof system; i.e., one in which, from proofs $(\pi_1, \ldots, \pi_n)$ that $(x_1, \ldots, x_n) \in L$, one can compute a proof π that $T_x(x_1, \ldots, x_n) \in L$, for an admissible transformation $T = (T_x, T_w)$:

Definition 2.3 (Malleable non-interactive proof system)**.** *Let* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *be a non-interactive proof system for a relation* R*. Let* $\mathcal{T}$ *be an allowable set of transformations for* R*. Then this proof system is* malleable with respect to $\mathcal{T}$ *if there exists an efficient algorithm* ZKEval *that on input* $(\sigma_{crs}, T, \{x_i, \pi_i\})$*, where* $T \in \mathcal{T}$ *is an* n*-ary transformation, and* $\mathcal{V}(\sigma_{crs}, x_i, \pi_i) = 1$ *for all* i*,* $1 \leq i \leq n$*, outputs a valid proof* π *for the statement* $x = T_x(\{x_i\})$ *(i.e., a proof* π *such that* $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$*).*

Going back to our above example, the algorithm ZKEval will take as input the transformation T (which is equivalent to taking as input the values a' and b'), and a proof π_1 that $x_1 = (G, g, A, B, C)$ is a DH tuple, and output a proof π that $x = T_x(x_1) = (G, g, A^{a'}, B^{b'}, C^{a'b'})$ is a DH tuple.

2.1 Derivation privacy for proofs

In addition to malleability, we must also consider a definition of derivation privacy analogous to the notion of function privacy for encryption, also called unlinkability [37], and defined formally in the next section. We have the following definition:

Definition 2.4 (Derivation privacy)**.** *For a non-interactive proof system* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *for an efficient relation* R *malleable with respect to* $\mathcal{T}$*, an adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $\sigma_{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$.
- *Step 2.* $(\mathsf{state}, x_1, w_1, \pi_1, \ldots, x_q, w_q, \pi_q, T) \xleftarrow{\$} \mathcal{A}(\sigma_{crs})$.
- *Step 3. If* $\mathcal{V}(\sigma_{crs}, x_i, \pi_i) = 0$ *for some* i*,* $(x_i, w_i) \notin R$ *for some* i*, or* $T \notin \mathcal{T}$*, abort and output* $\perp$*. Otherwise, form*

$$\pi \xleftarrow{\$} \begin{cases} \mathcal{P}(\sigma_{crs}, T_x(x_1, \ldots, x_q), T_w(w_1, \ldots, w_q)) & \text{if } b = 0 \\ \mathsf{ZKEval}(\sigma_{crs}, T, \{x_i, \pi_i\}) & \text{if } b = 1. \end{cases}$$

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \pi)$.

We say that the proof system is derivation private *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

In some cases, we would like to work with a stronger definition that applies only for NIZKs. In this case, the adversary will not be asked to provide witnesses or distinguish between the outputs of the prover and ZKEval, but instead between the zero-knowledge simulator and ZKEval. It will also be given the simulation trapdoor so that it can generate its own simulated proofs.

Definition 2.5 (Strong derivation privacy)**.** *For a malleable NIZK* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *with an associated simulator* (S_1, S_2)*, a given adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(\sigma_{sim}, \tau_s) \xleftarrow{\$} S_1(1^k)$.
- *Step 2.* $(\mathsf{state}, x_1, \pi_1, \ldots, x_q, \pi_q, T) \xleftarrow{\$} \mathcal{A}(\sigma_{sim}, \tau_s)$.
- *Step 3. If* $\mathcal{V}(\sigma_{sim}, x_i, \pi_i) = 0$ *for some* i*,* $(x_1, \ldots, x_q)$ *is not in the domain of* T_x*, or* $T \notin \mathcal{T}$*, abort and output* $\perp$*. Otherwise, form*

$$\pi \xleftarrow{\$} \begin{cases} S_2(\sigma_{sim}, \tau_s, T_x(x_1, \ldots, x_q)) & \text{if } b = 0 \\ \mathsf{ZKEval}(\sigma_{sim}, T, \{x_i, \pi_i\}) & \text{if } b = 1. \end{cases}$$

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \pi)$.

We say that the proof system is strongly derivation private *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

As we will see in Section 3, schemes that satisfy the weaker notion of derivation privacy can in fact be generically "boosted" to obtain schemes that satisfy the stronger notion. We can also see a generic way to obtain derivation privacy using malleability and the notion of *randomizability* for proofs, defined by Belenkiy et al. [8]as follows:

Definition 2.6 (Randomizable non-interactive proof system)**. [8]** *For a proof system* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *with an additional randomized algorithm* $\mathsf{RandProof}$ *that, on input a proof and a statement outputs a new proof for the same statement, a given adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $\sigma_{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$.
- *Step 2.* $(\mathsf{state}, x, w, \pi) \xleftarrow{\$} \mathcal{A}(\sigma_{crs})$.
- *Step 3. If* $\mathcal{V}(\sigma_{crs}, x, \pi) \neq 0$ *or* $(x, w) \notin R$ *then output* $\perp$*. Otherwise form*

$$\pi' \xleftarrow{\$} \begin{cases} \mathcal{P}(\sigma_{crs}, x, w) & \text{if } b = 0 \\ \mathsf{RandProof}(\sigma_{crs}, x, \pi) & \text{if } b = 1. \end{cases}$$

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \pi')$.

We say that the proof system is randomizable *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

Theorem 2.7. *If a proof system is both malleable and randomizable and uses* $\mathsf{ZKEval}' := \mathsf{RandProof} \circ \mathsf{ZKEval}$*, then it is also derivation private.*

Proof. To show this, we can take an adversary $\mathcal{A}$ that breaks the derivation privacy of the proof system with non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that wins at the randomizability game with related non-negligible advantage ϵ'. As input, $\mathcal{B}$ will first receive some CRS σ_{crs}, which it then passes along to $\mathcal{A}$; $\mathcal{B}$ will then receive in turn a tuple $(\{x_i\}, \{w_i\}, \{\pi_i\}, T)$. It can now perform the checks that $(x_i, w_i) \in R$ for all i, $T \in \mathcal{T}$, and $\mathcal{V}(\sigma_{\text{crs}}, x_i, \pi_i) = 1$ for all i; if any of these checks fails, then $\mathcal{A}$ was not successful and $\mathcal{B}$ can abort. Otherwise, it can now compute $x := T_x(\{x_i\})$, $w := T_w(\{w_i\})$, and $\pi \xleftarrow{\$} \mathsf{ZKEval}(\sigma_{\text{crs}}, T, \{x_i, \pi_i\})$, and output the tuple (x, w, π) as its challenge. It then gets back a proof π' and can give this to $\mathcal{A}$; it then outputs the same guess bit at the end of the game.

To see that interactions with $\mathcal{B}$ are indistinguishable from those that $\mathcal{A}$ would expect in the ordinary game, we first notice that the σ_{crs} given to $\mathcal{A}$ is identical to the honest one, so that we need only focus on the value π'. In the case that $b = 0$, we know that $\mathcal{B}$ will get back and give to $\mathcal{A}$ $\pi' \xleftarrow{\$} \mathcal{P}(\sigma_{\text{crs}}, x, w) = \mathcal{P}(\sigma_{\text{crs}}, T_x(\{x_i\}), T_w(\{w_i\}))$ which is exactly what $\mathcal{A}$ was expecting; similarly, if $b = 1$, then $\pi' \xleftarrow{\$} \mathsf{RandProof}(\sigma_{\text{crs}}, x, \pi) = \mathsf{RandProof}(\sigma_{\text{crs}}, x, \mathsf{ZKEval}(\sigma_{\text{crs}}, T, \{x_i, \pi_i\})) = \mathsf{ZKEval}'(\sigma_{\text{crs}}, T, \{x_i, \pi_i\})$, which is again exactly what $\mathcal{A}$ was expecting. $\mathcal{A}$ will therefore have the same advantage interacting with $\mathcal{B}$ as it does in the honest game, and as $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does we can conclude that $\mathcal{B}$ will succeed with non-negligible advantage as well. □

2.2 Function privacy for encryption

In this section, we highlight similarities between our newly introduced notions of malleability for proof systems, and notions of malleability for encryption, and we prove a theorem for encryption that is analogous to Theorem 2.7 above.

Recall that a proof can be randomizable, so that from an existing proof for a statement one can derive a new proof of the same statement. Similarly, an encryption scheme can be re-randomizable, so that from one ciphertext for a message m one can obtain another random ciphertext for the same message. We also defined derivation privacy for malleable proofs: a proof that is obtained from another proof is indistinguishable from one that is computed just from a witness. The analogous notion for encryption is *function privacy*, where from a given ciphertext, it is hard to tell whether this is a ciphertext computed by the encryption algorithm, or one that is obtained from another ciphertext via some homomorphic operations.

We just showed, in the previous section, that, in order to obtain derivation privacy for proof systems it is sufficient to have a malleable proof system that is also randomizable. Here, we show that, in order to achieve function privacy for encryption, it is sufficient to have a malleable (homomorphic) cryptosystem that is also re-randomizable.

We begin by giving a definition of re-randomizable encryption. Our definition here follows both our own analogous definition of randomizable proofs from the previous section, as well as the outline of the one due to Prabhakaran and Rosulek [36]; the similarities between these definitional approaches are important motivation for the definitions in the previous section.

Definition 2.8 (Re-randomizable encryption)**.** *For an encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *with an additional randomized algorithm* ReRand *that, on input a ciphertext outputs a new ciphertext with the same plaintext, a given adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event* $b' = 0$ *in the following game:*

- *Step 1.* $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$.
- *Step 2.* $(\mathsf{state}, c) \xleftarrow{\$} \mathcal{A}(pk, sk)$.

- *Step 3.* $c' \xleftarrow{\$} \begin{cases} \mathsf{Enc}(pk, \mathsf{Dec}(sk, c)) & \text{if } b = 0 \\ \mathsf{ReRand}(pk, c) & \text{if } b = 1. \end{cases}$

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, c')$.

We say that the encryption scheme is re-randomizable *if for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.*

Before we define function privacy (as modified from the definition of unlinkability due to Prabhakaran and Rosulek [37]), we need to formalize what we mean by a homomorphic encryption scheme. We again keep our notation consistent with what we use for proofs: transformations are defined as a pair (T_c, T_m) corresponding to respective transformations on ciphertexts and messages, and a *homomorphic* encryption scheme is an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with an additional algorithm Eval that, on input $(pk, \{c_i\}, T)$, outputs a ciphertext c such that, if the values contained in c_i is called m_i, $\mathsf{Dec}(sk, c) = T_m(\{m_i\})$.

Unlike our transformations for proofs, we allow our transformations T_m to act even on $\perp$ (i.e., we would like Eval to work even on ciphertexts that are malformed). We therefore require that $T_m(m_1, \ldots, m_n) = \perp$ if $m_i = \perp$ for any i, and similarly that $\mathsf{Eval}(pk, \{c_i\}, T) = \perp$ if $\mathsf{Dec}(sk, c_i) = \perp$ for any i. (Note that this is in keeping with how transformations are defined by Prabhakaran and Rosulek.)

Definition 2.9 (Function privacy)**.** *For a homomorphic encryption scheme* $(\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Dec})$*, a given adversary $\mathcal{A}$, and a bit b, let $p_b^{\mathcal{A}}(k)$ be the probability of the event $b' = 0$ in the following game:*

- *Step 1.* $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$.

- *Step 2.* $(\mathsf{state}, \{c_i\}, T) \xleftarrow{\$} \mathcal{A}(pk, sk)$.

- *Step 3. If $T \notin \mathcal{T}$ then abort. Otherwise, compute*

$$c' \xleftarrow{\$} \begin{cases} \mathsf{Enc}(pk, T_m(\{\mathsf{Dec}(sk, c_i)\})) & \text{if } b = 0 \\ \mathsf{Eval}(pk, \{c_i\}, T) & \text{if } b = 1. \end{cases}$$

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, c')$.

We say that the encryption scheme is function private *with respect to $\mathcal{T}$ if for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.*

We are now ready to state and prove our theorem. Note that, although no formal statement and proof of this theorem exists in the literature (as far as we know), this was previously known and used in the literature on fully homomorphic encryption [27, 41].

Theorem 2.10. *If an encryption scheme is both homomorphic and re-randomizable and uses* $\mathsf{Eval}' := \mathsf{ReRand} \circ \mathsf{Eval}$*, then it is function private.*

Proof. To show this, we can take an adversary $\mathcal{A}$ that breaks the function privacy of the encryption scheme with non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that wins at the re-randomizability game with related non-negligible advantage ϵ'. As input, $\mathcal{B}$ will first receive some keypair (pk, sk), which it then passes along to $\mathcal{A}$; $\mathcal{B}$ will then receive in turn a tuple $(\{c_i\}, T)$.

If $T \in \mathcal{T}$ then it can now compute $c' := \mathsf{Eval}(pk, \{c_i\}, T)$ (otherwise it will just abort), and output c' as its challenge. It then gets back a ciphertext c'' and can give this to $\mathcal{A}$; it then outputs the same guess bit at the end of the game.

To see that interactions with $\mathcal{B}$ are indistinguishable from those that $\mathcal{A}$ would expect in the ordinary game, we first notice that the (pk, sk) given to $\mathcal{A}$ is identical to the honest one, so that we need only focus on the value c''. In the case that $b = 0$, we know that $\mathcal{B}$ will get back and give to $\mathcal{A}$ $c'' \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, c')) = \mathsf{Enc}(pk, T_m(\{m_i\}))$ which is exactly what $\mathcal{A}$ was expecting; similarly, if $b = 1$, then $c'' \xleftarrow{\$} \mathsf{ReRand}(pk, c') = \mathsf{ReRand}(pk, \mathsf{Eval}(pk, \{c_i\}, T)) = \mathsf{Eval}'(pk, \{c_i\}, T)$, which is again exactly what $\mathcal{A}$ was expecting. $\mathcal{A}$ will therefore have the same advantage interacting with $\mathcal{B}$ as it does in the honest game, and as $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does we can conclude that $\mathcal{B}$ will succeed with non-negligible advantage as well. □

3 Controlled Malleability for NIZKs

Is the notion of malleability compatible with the notion of a proof of knowledge or with strong notions like simulation soundness? Recall that to achieve simulation soundness, as defined by Sahai and de Santis et al. [38, 19], we intuitively want an adversary $\mathcal{A}$ to be unable to produce a proof of a new false statement even if it can request many such proofs from the simulator; for the even stronger notion of simulation-extractability as defined by de Santis et al. and Groth [19, 31], a proof system must admit an efficient extractor that finds witnesses to all statements proved by an adversary, again even when the adversary has access to a simulator.

Malleability, in contrast, explicitly allows an adversary to take as input the values x', π', apply some admissible transformation T to x' to obtain $x = T_x(x')$, and compute a proof π that $x \in L_R$; importantly, the adversary can do all this without knowing the original witness w'. Suppose, for a malleable proof system, that the adversary is given as input a *simulated* proof π' that was generated without access to the witness w' for x', and for concreteness let T be the identity transformation. Then requiring that, on input (x, π), the extractor should output w, implies that membership in L_R can be tested for a given x by computing a simulated proof, mauling it, and then extracting the witness from the resulting proof (formally, this would mean that $L_R \in \mathbf{RP}$). Thus, seemingly, one cannot reconcile the notion of malleability with that of a simulation-extractable proof of knowledge.

Surprisingly, however, under a relaxed but still meaningful extractability requirement, we can have a proof system that is both malleable and simulation-extractable to a satisfactory extent; we call this notion *controlled malleability*. Essentially this definition will require that the extractor can extract either a valid witness, or a previously proved statement x' and a transformation T in our allowed set $\mathcal{T}$ that could be used to transform x' into the new statement x. To demonstrate that our definition is useful, we will show in Section 5 that it can be used to realize a strong notion of encryption security, and in Section 6 that it can also be used to reduce the overall size of proofs for verifiable shuffles.

Definition 3.1 (Controlled-malleable simulation sound extractability)**.** *Let* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *be a NIZKPoK system for an efficient relation* R*, with a simulator* (S_1, S_2) *and an extractor* (E_1, E_2)*. Let* $\mathcal{T}$ *be an allowable set of unary transformations for the relation* R *such that membership in* $\mathcal{T}$ *is efficiently testable. Let* SE_1 *be an algorithm that, on input* 1^k *outputs* $(\sigma_{crs}, \tau_s, \tau_e)$ *such that* (σ_{crs}, τ_s) *is distributed identically to the output of* S_1*. Let* $\mathcal{A}$ *be given, and consider the following game:*

- *Step 1.* $(\sigma_{crs}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$.

- *Step 2.* $(x, \pi) \xleftarrow{\$} \mathcal{A}^{S_2(\sigma_{crs}, \tau_s, \cdot)}(\sigma_{crs}, \tau_e)$.
- *Step 3.* $(w, x', T) \leftarrow E_2(\sigma_{crs}, \tau_e, x, \pi)$.

We say that the NIZKPoK satisfies controlled-malleable simulation-sound extractability *(CM-SSE, for short) if for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability (over the choices of SE_1, $\mathcal{A}$, and S_2) that $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$ and $(x, \pi) \notin Q$ (where Q is the set of queried statements and their responses) but either (1) $w \neq \perp$ and $(x, w) \notin R$; (2) $(x', T) \neq (\perp, \perp)$ and either $x' \notin Q_x$ (the set of queried instances), $x \neq T_x(x')$, or $T \notin \mathcal{T}$; or (3) $(w, x', T) = (\perp, \perp, \perp)$ is at most $\nu(k)$.*

This definition is actually closely related to simulation-extractability; in fact, if we restrict our set of transformations to be $\mathcal{T} = \emptyset$, we obtain exactly Groth's notion of simulation-sound extractability. Note also that this definition does not require that a proof system actually be malleable, it only requires that, should it happen to be malleable, this malleability be limited in a controlled way. Thus, a simulation-sound extractable proof system would also satisfy our definition, for any set $\mathcal{T}$, even though it is not malleable. We refer to a proof system that is both strongly derivation private and controlled-malleable simulation-sound extractable as a *controlled-malleable NIZK* (cm-NIZK).

Finally, note that our definition applies only to unary transformations. This is because our requirement that we can extract the transformation T means we cannot hope to construct cm-NIZKs for n-ary transformations where $n > 1$, as this would seem to necessarily expand the size of the proof (similarly to what Prabhakaran and Rosulek show for HCCA encryption [37]). We therefore achieve cm-NIZKs for classes of unary transformations that are closed under composition (i.e., $T' \circ T \in \mathcal{T}$ for all $T, T' \in \mathcal{T}$). In addition, our simulation strategy depends on the identity transformation being a member of $\mathcal{T}$, so we can achieve cm-NIZKs only for classes of transformations that include the identity transformation.

A generic construction

Let R be an efficient relation, and suppose $\mathcal{T}$ is an allowable set of transformations for R that contains the identity transformation; suppose further that membership in $\mathcal{T}$ is efficiently testable. Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a secure signature scheme. Let $(\mathsf{CRSSetup}_{\mathsf{WI}}, \mathcal{P}_{\mathsf{WI}}, \mathcal{V}_{\mathsf{WI}})$ be a NIWIPoK for the following relation R_{WI}: $((x, vk), (w, x', T, \sigma)) \in R_{\mathsf{WI}}$ if $(x, w) \in R$ or $\mathsf{Verify}(vk, \sigma, x') = 1$, $x = T_x(x')$, and $T \in \mathcal{T}$. Consider the proof system $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ defined as follows:

- $\mathsf{CRSSetup}(1^k)$: First generate $\sigma_{\mathsf{WI}crs} \xleftarrow{\$} \mathsf{CRSSetup}_{\mathsf{WI}}(1^k)$ and $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; then output $\sigma_{\text{crs}} := (\sigma_{\mathsf{WI}crs}, vk)$.
- $\mathcal{P}(\sigma_{\text{crs}}, x, w)$: Output $\pi \xleftarrow{\$} \mathcal{P}_{\mathsf{WI}}(\sigma_{\mathsf{WI}crs}, x_{\mathsf{WI}}, w_{\mathsf{WI}})$, where $x_{\mathsf{WI}} = (x, vk)$ and $w_{\mathsf{WI}} = (w, \perp, \perp, \perp)$.
- $\mathcal{V}(\sigma_{\text{crs}}, x, \pi)$: Output $\mathcal{V}_{\mathsf{WI}}(\sigma_{\mathsf{WI}crs}, x_{\mathsf{WI}}, \pi)$ where $x_{\mathsf{WI}} = (x, vk)$.

To obtain strong derivation privacy with respect to R and $\mathcal{T}$ we also require the NIWIPoK to be derivation private with respect to R_{WI} and a set of transformations $\mathcal{T}_{\mathsf{WI}}$ such that for every $T' = (T'_x, T'_w) \in \mathcal{T}$ there exists a $T_{\mathsf{WI}}(T') \in \mathcal{T}_{\mathsf{WI}}$. For $T_{\mathsf{WI}}(T') = (T_{\mathsf{WI},x}, T_{\mathsf{WI},w})$ we require that $T_{\mathsf{WI},x}(x, vk) = (T'_x(x), vk)$, and $T_{\mathsf{WI},w}(w, x', T, \sigma) = (T'_w(w), x', T' \circ T, \sigma)$. Assuming our underlying NIWI is malleable, we can define ZKEval in terms of $\mathsf{ZKEval}_{\mathsf{WI}}$:

- $\mathsf{ZKEval}(\sigma_{\text{crs}}, T, x, \pi)$: Output $\mathsf{ZKEval}_{\mathsf{WI}}(\sigma_{\mathsf{WI}crs}, T_{\mathsf{WI}}(T), x_{\mathsf{WI}}, \pi)$ where $x_{\mathsf{WI}} = (x, vk)$.

To see that this construction gives us the desired properties, we have the following three theorems:

Theorem 3.2. *If the underlying non-interactive proof system is witness indistinguishable, the scheme described above is zero knowledge.*

Proof. To show this, we describe a simulator (S_1, S_2) for the NIZK such that an adversary that distinguishes between the outputs of S_1 and S_2 and the honest values used for the CRS and proofs with some non-negligible advantage would be able to distinguish either between an honest and a simulated CRS or between witnesses in the underlying NIWIPoK with the same advantage.

The simulator S_1 will first use the honest setup algorithm to generate $\sigma_{\mathsf{WI}crs}$; it will also generate a signing keypair (vk, sk) and use $\tau_s := sk$. Now, when S_2 is asked to provide a proof for a statement x, it will compute $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, x)$, honestly prove knowledge of the tuple $(\bot, x, \mathrm{id}, \sigma)$, and return the generated proof π.

As the signing keypair was generated honestly, the whole σ_{crs} returned by S_1 will be identical to an honest one. that the proofs will be indistinguishable as well, we switch to the game that uses S_2 instead of the prover. As S_2 is providing a valid witness and the proof system is witness indistinguishable the output of S_2 will be indistinguishable from the output of an honest prover. □

Theorem 3.3. *If the underlying signature scheme is EUF-CMA secure and the underlying NIWIPoK is extractable, the scheme described above satisfies controlled-malleable simulation-sound extractability.*

Proof. To demonstrate that the NIZK is CM-SSE, we can use the same simulator S_2 from the proof above; that is, a simulator with access to the signing key sk that forms signatures and thus only ever uses the second type of witness. For the extraction trapdoor, we will use the τ_e for the underlying PoK, so that SE_1 will first run E_1 to generate an honest $\sigma_{\mathsf{WI}crs}$ along with the trapdoor τ_e, then generate a signing keypair (vk, sk), and finally set $\sigma_{\mathrm{crs}} := (\sigma_{\mathsf{WI}crs}, vk)$ and $\tau_s := sk$.[5] Now, given this $(\sigma_{\mathrm{crs}}, \tau_e)$ and access to S_2, an adversary $\mathcal{A}$ will play the CM-SSE game and eventually output the pair (x, π), from which we will extract a tuple (w, x', T, σ). In order for $\mathcal{A}$ to be considered successful, recall that there are five cases:

1. $w \neq \bot$ and $(x, w) \notin R$, or
2. $(x', T) \neq (\bot, \bot)$ and $\mathsf{Verify}(vk, \sigma, x') = 1$ but $\mathcal{A}$ never queried S_2 on x', or
3. $(x', T) \neq (\bot, \bot)$ and $\mathsf{Verify}(vk, \sigma, x') = 1$ but $x \neq T(x')$, or
4. $(x', T) \neq (\bot, \bot)$ and $\mathsf{Verify}(vk, \sigma, x') = 1$, $x = T(x')$, but $T \notin \mathcal{T}$, or
5. $(w, x', T) = (\bot, \bot, \bot)$.

For all but the second item, the extractability of the underlying proof of knowledge implies that this case cannot hold, as it would imply that $\mathcal{A}$ had output a proof for which the extractor could not come up with a valid witness, and thus violate the underlying extractability property. For the second case, the unforgeability of the signature scheme implies that it cannot happen either, as

[5]Note that, because of the requirement that the restriction to $(\sigma_{\mathrm{crs}}, \tau_s)$ be identical to that output by S_1, we seemingly require the underlying NIWIPoK to be perfectly extractable (as our simulator from the previous proof uses an honest CRS). If we change the simulator to be (SE_1, S_2), however, then we can still achieve zero knowledge and furthermore easily satisfy the requirement that the distributions be identical; alternatively, we could alter the definition to require only that the output of SE_1 restricted to $(\sigma_{\mathrm{crs}}, \tau_s)$ be indistinguishable from the output of S_1, rather than identical.

it requires $\mathcal{A}$ to come up with a signature σ for a new statement x' on its own. Therefore, the unforgeability of the signature scheme and the extractability of the proof of knowledge together imply that none of the above cases can occur with more than negligible probability, and so we are done. □

Theorem 3.4. *If the underlying NIWIPoK is derivation private for $\mathcal{T}_{\mathsf{WI}}$ (as defined in Definition 2.4), then the scheme described above is strongly derivation private for $\mathcal{T}$ (as defined in Definition 2.5).*

Proof. To show this, we can take an adversary $\mathcal{A}$ that breaks strong derivation privacy for our cm-NIZK with some non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that breaks derivation privacy for the NIWIPoK with the same advantage ϵ. To start, $\mathcal{B}$ will receive $\sigma_{\mathsf{WI}crs}$ for the NIWIPoK. It then generates $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$, keeps sk and gives $\sigma_{\text{crs}} = (\sigma_{\mathsf{WI}crs}, vk)$ to $\mathcal{A}$; note that this means $\mathcal{B}$ can now act as the simulator S_2 from the proof of Theorem 3.2. On S_2 oracle queries then, $\mathcal{B}$ will just run the same code as S_2: namely, on queries x it will use sk to compute $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, x)$, and then form and return to $\mathcal{A}$ a proof $\pi \xleftarrow{\$} \mathcal{P}_{\mathsf{WI}}(\sigma_{\mathsf{WI}crs}, (x, vk), (\perp, x, \text{id}, \sigma))$. Now, on $\mathcal{A}$'s challenge query $(\{x_i, \pi_i\}, T)$, $\mathcal{B}$ can use the same trick again to form witnesses $w_{\mathsf{WI}i}$ as $w_{\mathsf{WI}i} := (\perp, x_i, \text{id}, \mathsf{Sign}(sk, x_i))$; it then queries its oracle on $(\{(x_i, vk), w_{\mathsf{WI}i}, \pi_i\}, T_{\mathsf{WI}}(T))$ to receive a proof π which it then passes along to $\mathcal{A}$. At the end of the game, $\mathcal{B}$ will output whatever guess bit $\mathcal{A}$ does.

To see that interactions with $\mathcal{B}$ are indistinguishable from interactions in the honest strong derivation privacy game, we first note that $\mathcal{B}$ exactly follows the code for both S_1 in the CRS generation and S_2 in the oracle queries, so that the interaction in both of these steps will be identical. As for the challenge query, we note that $\mathcal{B}$ is again computing the same valid witnesses for the underlying proof as S_2 would. If its query is answered by $\mathcal{P}_{\mathsf{WI}}$ then, the proof it returns to $\mathcal{A}$ is distributed identically to a proof from S_2, while if the query is answered by $\mathsf{ZKEval}_{\mathsf{WI}}$ it is trivially distributed identically to a proof from ZKEval. We therefore have that the success cases for $\mathcal{A}$ and $\mathcal{B}$ correspond exactly, and thus $\mathcal{B}$ will succeed whenever $\mathcal{A}$ does. □

In addition, we would like to ensure that this construction can in fact be instantiated efficiently for many useful sets $\mathcal{T}$ with a derivation-private NIWIPoK; it turns out that this can be done by combining Groth-Sahai proofs [34] with a special type of signature called a *structure-preserving signature*. For more details, we defer to Section 4.2.

4 Instantiating cm-NIZKs Using Groth-Sahai Proofs

In this section, we explore the malleability of Groth-Sahai (GS) proofs [34]. This will allow us to efficiently instantiate controlled-malleable proofs for a large class of transformations.

4.1 Malleability for Groth-Sahai Proofs

We aim to fully characterize the class of transformations with respect to which GS proofs can be made malleable. First, we recall that GS proofs allow a prover to prove knowledge of a satisfying assignment to a list of (homogeneous) *pairing product equations* eq of the form $\prod_{i,j\in[1..n]} e(x_i, x_j)^{\gamma_{ij}} = 1$ concerning the set of variables $x_1, \ldots, x_n \in \mathbb{G}$. Furthermore, some of the variables in these equations may be fixed to be specific constant values (for example, the public group generator g). In what follows we will use $a, b, c, \ldots$ to denote fixed constants, and $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$ to denote unconstrained variables. An instance x of such a *pairing product statement* consists of the list of equations

$\mathsf{eq}_1, \ldots, \mathsf{eq}_\ell$ (fully described by their exponents $\{\gamma_{ij}^{(1)}\}, \ldots, \{\gamma_{ij}^{(\ell)}\}$) and the values of the constrained variables (fully described by the list $a_1, \ldots, a_{n'} \in \mathbb{G}$ for $n' \leq n$).

In the existing literature, there are already various examples [20, 7, 24] of ways in which pairing product statements and the accompanying Groth-Sahai proofs can be mauled. Here, we attempt to generalize these previous works by providing a characterization of *all* the ways in which GS proofs of pairing product statements can be mauled; we then show, in Appendix A.4, how these previous examples can be obtained as special cases of our general characterization.

To start, we describe transformations on pairing product instances in terms of a few basic operations. We will say that any transformation that can be described as a composition of these operations is a *valid transformation*. For each valid transformation we show, in Appendix A.2, that there is a corresponding ZKEval procedure that updates the GS proof to prove the new statement. Finally, we present in Appendix A.5some other convenient operations that can be derived from our minimal set.

To help illustrate the usage of our basic transformations, we will consider their effect on the pairing product instance $(\mathsf{eq}_1, \mathsf{eq}_2, a, b)$, where $\mathsf{eq}_1 := e(\mathbf{x}, b)e(a, b) = 1$ and $\mathsf{eq}_2 := e(a, \mathbf{y}) = 1$. Note that here we will describe the transformations in terms of their effect on the instances, but in all of these operations the corresponding witness transformations T_w are easily derived from the instance transformations T_x.

Definition 4.1. *(Informal) A* valid transformation *is one that can be expressed as some combination of (a polynomial number of) the following six operations:*

1. *Merge equations:* $\mathsf{MergeEq}(\mathsf{eq}_i, \mathsf{eq}_j)$ *adds the product of* eq_i *and* eq_j *as a new equation.*

   Ex. $\mathsf{MergeEq}(\mathsf{eq}_1, \mathsf{eq}_2)$ *adds the equation* $e(\mathbf{x}, b)e(a, b)e(a, \mathbf{y}) = 1$

2. *Merge variables:* $\mathsf{MergeVar}(x, y, z, S)$ *generates a new variable* z*. If* x *and* y *are both constants,* z *will have value* xy*. Otherwise* $\mathbf{z}$ *will be unconstrained. For every variable* w *in the set* S*, we add the equation* $e(xy, w)^{-1}e(z, w) = 1$.[6]

   Ex. $\mathsf{MergeVar}(x, a, z, \{x, b, z\})$ *adds the variable* z *and the equations*

   $$e(\mathbf{x}a, \mathbf{x})^{-1}e(\mathbf{z}, \mathbf{x}) = 1, \quad e(\mathbf{x}a, b)^{-1}e(\mathbf{z}, b) = 1, \quad \text{and } e(\mathbf{x}a, \mathbf{z})^{-1}e(\mathbf{z}, \mathbf{z}) = 1.$$

3. *Exponentiate variable:* $\mathsf{ExpVar}(x, \delta, z, S)$ *generates a new variable* z*. If* x *is a constant,* $z = x^\delta$*, otherwise it will be unconstrained. For every variable* $w \in S$*, we add the equation* $e(x, w)^{-\delta}e(z, w) = 1$.

   Ex. $\mathsf{ExpVar}(x, \delta, z, \{x, b, z\})$ *adds the variable* z *and the equations*

   $$e(\mathbf{x}, \mathbf{x})^{-\delta}e(\mathbf{z}, \mathbf{x}) = 1, \quad e(\mathbf{x}, b)^{-\delta}e(\mathbf{z}, b) = 1, \quad \text{and } e(\mathbf{x}, \mathbf{z})^{-\delta}e(\mathbf{z}, \mathbf{z}) = 1.$$

4. *Add constant equation:* $\mathsf{Add}(\{a_i\}, \{b_j\}, \{\gamma_{ij}\})$ *takes a set of constants* a_i, b_i*, satisfying a pairing product equation* $\prod e(a_i, b_j)^{\gamma_{ij}} = 1$ *and adds these variables and the new equation to the statement.*

   Ex. $\mathsf{Add}(\{g\}, \{1\}, \{1\})$ *adds the variables* $g, 1$ *and the equation* $\mathsf{eq}_3 := e(g, 1) = 1$.

   We often write as a shorthand $\mathsf{Add}(\mathsf{eq}_3 := e(g, 1) = 1)$.

5. *Remove equation:* $\mathsf{RemoveEq}(\mathsf{eq}_i)$ *simply removes equation* eq_i *from the list.*

   Ex. $\mathsf{RemoveEq}(\mathsf{eq}_2)$ *removes the equation* $e(a, \mathbf{y}) = 1$ *from the equation list.*

[6] This is shorthand for $e(x, w)^{-1}e(y, w)^{-1}e(z, w) = 1$.

6. *Remove variable:* $\mathsf{RemoveVar}(x)$ *removes the variable* x *from the variable set iff* x *does not appear in any of the listed equations.*

   Ex. We cannot remove any of the variables from the example statement. However, we could do $\mathsf{RemoveEq}(\mathsf{eq}_2)$ *and then* $\mathsf{RemoveVar}(y)$*, which would remove the equation* $e(a, \mathbf{y}) = 1$ *from the equation list and the variable* y *from the set of variables.*

A proof of the following lemma appears in Appendix A.2:

Lemma 4.2. *There exists an efficient procedure* ZKEval *such that given any pairing product instance* x*, any valid transformation* T*, and any accepting Groth-Sahai proof* π *for* x*,* $\mathsf{ZKEval}(x, \pi, T)$ *produces an accepting proof for* $T(x)$*.*

4.2 An efficient instantiation of controlled malleable NIZKs

Looking back at Section 3 we see that there are two main components needed to efficiently instantiate a controlled-malleable NIZK proof system: appropriately malleable proofs and signatures that can be used in conjunction with these proofs.

First we consider the set of relations and tranformations for which we can use Groth-Sahai proofs to construct the necessary malleable NIWIPoKs.

Definition 4.3. *For a relation* R *and a class of transformations* $\mathcal{T}$*, we say* $(R, \mathcal{T})$ *is* CM-friendly *if the following six properties hold: (1) representable statements: any instance and witness of* R *can be represented as a set of group elements; (2) representable transformations: any transformation in* $\mathcal{T}$ *can be represented as a set of group elements; (3) provable statements: we can prove the statement* $(x, w) \in R$ *using pairing product equations; (4) provable transformations: we can prove the statement "*$T_x(x') = x$ *for* $T \in \mathcal{T}$*" using pairing product equations; (5) transformable statements: for any* $T \in \mathcal{T}$ *there is a valid transformation from the statement "*$(x, w) \in R$*" to the statement "*$(T_x(x), T_w(w)) \in R$*"; and (6) transformable transformations: for any* $T, T' \in \mathcal{T}$ *there is a valid transformation from the statement "*$T_x(x') = x$ *for* $T = (T_x, T_w) \in \mathcal{T}$*" to the statement "*$T'_x \circ T_x(x') = T'_x(x)$ *for* $T' \circ T \in \mathcal{T}$*."*

In order for the signatures to be used within our construction, we know that they need to have pairing-based verifiability (i.e., we can represent the Verify algorithm in terms of a set of GS equations), and that the values being signed are group elements, so that they can be efficiently extracted from the proof (as GS proofs are extractable for group elements only, not exponents). These requirements seem to imply the need for *structure-preserving signatures* [3], which we can define for the symmetric setting as follows:

Definition 4.4. *A signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *over a bilinear group* (p, G, G_T, g, e) *is said to be* structure preserving *if the verification key, messages, and signatures all consist of group elements in* G*, and the verification algorithm evaluates membership in* G *and pairing product equations.*

Since their introduction, three structure-preserving signature schemes have emerged that would be suitable for our purposes; all three have advantages and disadvantages. The first, due to Abe, Haralambiev, and Ohkubo [5, 3] is quite efficient but uses a slightly strong q-type assumption. The second, due to Abe et al. [4], is optimally efficient but provably secure only in the generic group model. The third and most recent, due to Chase and Kohlweiss [15], is significantly less efficient than the previous two, but relies for its security on Decision Linear (DLIN) [11], which is already a relatively well-established assumption.

Because we can also instantiate GS proofs using DLIN, we focus on this last structure-preserving signature, keeping in mind that others may be substituted in for the sake of efficiency (but at the cost of adding an assumption). Putting these signatures and GS proofs together, we can show our main result of this section: given any CM-friendly relation and set of transformations $(R, \mathcal{T})$, we can combine structure-preserving signatures and malleable proofs to obtain a cm-NIZK. This can be stated as the following theorem (a proof of which can be found in Appendix C):

Theorem 4.5. *Given a derivation private NIWIPoK for pairing product statements that is malleable for the set of all valid transformations, and a structure preserving signature scheme, we can construct a cm-NIZK for any CM-friendly relation and transformation set* $(R, \mathcal{T})$*.*

In Appendix A.2, we show that Groth-Sahai proofs are malleable for the set of all valid transformations (as outlined in Definition 4.1) and that valid transformations can be carried through conjunctions and disjunctions. As Groth-Sahai proofs and structure-preserving signatures can both be constructed based on DLIN, we obtain the following theorem:

Theorem 4.6. *If DLIN holds, then we can construct a cm-NIZK that satisfies strong derivation privacy for any CM-friendly relation and transformation set* $(R, \mathcal{T})$*.*

5 Controlled Malleability for Encryption

As we mentioned earlier, malleability can also be an attractive feature for a cryptosystem: it allows computation on encrypted data. On the other hand, it seems to be in conflict with security: if a ciphertext can be transformed into a ciphertext for a related message, then the encryption scheme is clearly not secure under an adaptive chosen ciphertext attack, which is the standard notion of security for encryption.

Prabhakaran and Rosulek [36, 37] were the first to define and realize a meaningful notion of security in this context. Specifically, they introduced re-randomizable CCA security (RCCA) [36] and homomorphic CCA security (HCCA) [37]. In a nutshell, their definition of security is given as a game between a challenger and an adversary; the adversary receives a public key and a challenge ciphertext and can query the challenger for decryptions of ciphertexts. The challenger's ciphertext c^* is either a valid encryption of some message, or a dummy ciphertext; in the former case, the challenger answers the decryption queries honestly; in the latter case, the challenger may decide that a decryption query is a "derivative" ciphertext computed from c^* using some transformation T; if this is an allowed transformation, the challenger responds with $T(m)$, else it rejects the query. The adversary wins if it correctly guesses whether its challenge ciphertext was meaningful.[7] Prabhakaran and Rosulek achieve their notion of security under the decisional Diffie-Hellman assumption using ad-hoc techniques reminiscent of the Cramer-Shoup [17] cryptosystem.

In this section, we show that controlled-malleable NIZKs can be used as a general tool for achieving RCCA and HCCA security. Our construction is more modular than that of Prabhakaran and Rosulek: we construct a controlled-malleable-CCA-secure encryption scheme generically from a semantically secure one and a cm-NIZK for an appropriate language; where controlled-malleable-CCA security is our own notion of security that is, in some sense, a generalization of RCCA security and also captures the security goals of HCCA security. We then show how our construction can be instantiated using Groth-Sahai proofs, under the DLIN assumption in groups with bilinear maps.

[7] A formal definition and more detailed explanation of their notion of homomorphic-CCA (HCCA) security can be found in Appendix E.

5.1 Definition of Controlled-Malleable CCA Security

Our definitional goals here are (1) to give a definition of controlled malleability for encryption that closely mirrors our definition of controlled malleability for proofs, and (2) to give a definition that can be easily related to previous notions such as CCA, RCCA, and HCCA. We call this notion of security *controlled-malleable CCA* (CM-CCA) security.

Following Prabhakaran and Rosulek [37], CM-CCA requires the existence of two algorithms, SimEnc and SimExt. SimEnc creates ciphertexts that are distributed indistinguishably from regular ciphertexts (those generated using the encryption algorithm Enc), but contain no information about the queried message; this is modeled by having SimEnc not take any message as input. SimExt allows the challenger to track "derivative" ciphertexts. That is to say, on input a ciphertext c, SimExt determines if it was obtained by transforming some ciphertext c' previously generated using SimEnc; if so, SimExt outputs the corresponding transformation T.

The game between the challenger and the adversary in the definition of security is somewhat different from that in the definition by Prabhakaran and Rosulek. Specifically, we do not have a single challenge ciphertext c^*; instead, the adversary has access to an encryption and decryption oracles. Intuitively, for our definition we would like to say that an adversary cannot distinguish between two worlds: the real world in which it is given access to honest encryption and decryption oracles, and an ideal world in which it is given access to an ideal encryption oracle (which outputs ciphertexts containing no information about the queried message) and a decryption oracle that outputs a special answer for ciphertexts derived from the ideal ciphertexts (by using SimExt to track such ciphertexts) and honestly decrypts otherwise.

Let us consider transformations more closely. Recall that, for proofs of language membership, a transformation $T \in \mathcal{T}$ consists of a pair of transformations (T_x, T_w), where T_x acts on the instances, and T_w on the witnesses. What is the analogue for ciphertexts? A legal transformation T_x on a ciphertext implies some legal transformation T_m on an underlying message and a corresponding transformation T_r on the underlying randomness. Thus, here we view transformations as tuples $T = (T_x, (T_m, T_r))$, where T_x acts on the ciphertexts, T_m acts on the plaintexts, and T_r acts on the randomness.

In Appendix E, we relate CM-CCA security to CCA, RCCA and HCCA security. Specifically, we show that (1) when the class of allowed transformation $\mathcal{T}$ is the empty set, CM-CCA implies regular CCA security; (2) when the class of allowed transformations is as follows: $T \in \mathcal{T}$ if $T = (T_x, (T_m, T_r))$ where T_m is the identity transformation, then CM-CCA security implies RCCA security; (3) in more general cases we show that it implies the notion of targeted malleability introduced by Boneh et al. [12]; in addition, we show that our notion satisfies the UC definition given by Prabhakaran and Rosulek, so that it captures the desired HCCA security goals, even if it does not satisfy their definition of HCCA security (which is in fact a stronger notion).

Finally, because our cm-NIZK is malleable only with respect to unary transformations, we inherit the limitation that our encryption scheme is malleable only with respect to unary transformations as well; as our security definition is closely related to HCCA security and Prabharakan and Rosulek in fact prove HCCA security (combined with unlinkability) is impossible with respect to binary transformations, this is perhaps not surprising.

Definition 5.1. *For an encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, a class of transformations* $\mathcal{T}$*, an adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event* $b' = 0$ *in the following game: first* $(pk, sk) \xleftarrow{\$} K(1^k)$*, and next* $b' \xleftarrow{\$} \mathcal{A}^{E_{pk}(\cdot), D_{sk}(\cdot)}(pk)$*, where* (K, E, D) *are defined as* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *if* $b = 0$*, and the following algorithms (defined for a state set* $Q = Q_m \times Q_c = \{(m_i, c_i)\}$*) if* $b = 1$*:*

Procedure $K(1^k)$	*Procedure* $E(pk, m)$	*Procedure* $D(sk, c)$
$(pk, sk, \tau_1, \tau_2) \xleftarrow{\$} \mathsf{SimKeyGen}(1^k)$	$c \xleftarrow{\$} \mathsf{SimEnc}(pk, \tau_1)$	$(c', T) \leftarrow \mathsf{SimExt}(sk, \tau_2, c)$
return pk	*add* (m, c) *to* Q	*if* $\exists i$ *s.t.* $c' = c_i \in Q_c$ *and* $T \neq \perp$ *return* $T_m(m_i)$
	return c	*else return* $\mathsf{Dec}(sk, c)$

We say that the encryption scheme is controlled-malleable-CCA secure *(or CM-CCA secure for short) if there exist PPT algorithms* SimKeyGen, SimEnc, *and* SimExt *as used above such that for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

5.2 A generic construction of CM-CCA-secure encryption

As mentioned earlier, we can obtain an encryption scheme that achieves this notion of security; we do this by combining a cm-NIZK $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ for the relation R such that $((pk, c), (m, r)) \in R$ iff $c := \mathsf{Enc}'(pk, m; r)$ and an IND-CPA-secure encryption scheme $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- $\mathsf{KeyGen}(1^k)$: Run $(pk', sk') \xleftarrow{\$} \mathsf{KeyGen}'(1^k)$ and $\sigma_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$; set $pk := (pk', \sigma_{\text{crs}})$ and $sk := (pk, sk')$ and output pk.
- $\mathsf{Enc}(pk, m)$: Parse $pk = (pk', \sigma_{\text{crs}})$; then compute $c' \xleftarrow{\$} \mathsf{Enc}'(pk', m)$ and $\pi \xleftarrow{\$} \mathcal{P}(\sigma_{\text{crs}}, (pk', c'), m)$ (i.e., a proof of knowledge of the value inside c') and output $c := (c', \pi)$.
- $\mathsf{Dec}(sk, c)$: First parse $sk = (pk, sk')$, $pk = (pk', \sigma_{\text{crs}})$, and $c = (c', \pi)$; now check that $\mathcal{V}(\sigma_{\text{crs}}, (pk', c'), \pi) = 1$. If not, abort and output $\perp$. Otherwise, compute and output $m = \mathsf{Dec}(sk', c')$.

Aside from having our encryption scheme be secure, we would also like the functionality of having the scheme be homomorphic with respect to the same operations as its underlying components. Assuming then that the underlying encryption scheme is homomorphic with respect to a class $\mathcal{T}$ and the underlying cm-NIZK is controlled malleable with respect to $\mathcal{T}$, we can define our Eval algorithm for $T \in \mathcal{T}$ in terms of the underlying Eval' and ZKEval algorithms as

- $\mathsf{Eval}(pk, c, T)$: Parse $pk = (pk', \sigma_{\text{crs}})$ and $c = (c', \pi)$; then compute $c'' \xleftarrow{\$} (\mathsf{Eval}(pk', c', T)$ and $\pi' \xleftarrow{\$} \mathsf{ZKEval}(\sigma_{\text{crs}}, T, (pk', c'), \pi)))$ and return (c'', π').

To show CM-CCA security, we define the algorithms SimKeyGen, SimEnc, and SimExt as follows:

- $\mathsf{SimKeyGen}(1^k)$: Run $(pk', sk') \xleftarrow{\$} \mathsf{KeyGen}'(1^k)$, $(\sigma_{\text{crs}}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$, and output $(pk := (pk', \sigma_{\text{crs}}), sk := (pk, sk'), \tau_1 := \tau_s, \tau_2 := \tau_e)$.
- $\mathsf{SimEnc}(pk, \tau_1)$: Pick a random message $r \xleftarrow{\$} \mathcal{M}$ and form $c' \xleftarrow{\$} \mathsf{Enc}(pk, r)$; form also a proof $\pi \xleftarrow{\$} S_2(\sigma_{\text{crs}}, \tau_1, (pk', c'))$ and return $c := (c', \pi)$.
- $\mathsf{SimExt}(sk, \tau_2, c)$: Parse $c = (c', \pi)$, $sk = (pk, sk')$, and $pk = (pk', \sigma_{\text{crs}})$; then check that $\mathcal{V}(\sigma_{\text{crs}}, (pk', c'), \pi) = 1$; if not, abort and return $(\perp, \perp)$. Otherwise, run $(w, x', T) = \mathsf{Extract}(\tau_2, \pi)$. If $(x', T) \neq (\perp, \perp)$ then parse $x' = (pk, c'')$ and check if $T \in \mathcal{T}$ and $T_c(c'') = c'$; if either of these checks fails then output $(\perp, \perp)$. Otherwise, output (c'', T).

An efficient instantiation of the above construction can be found in Appendix D.2; briefly, we use our cm-NIZK construction based on GS proofs [34] and Boneh-Boyen-Shacham (BBS) encryption [11]. As our concrete class of transformations we consider vector multiplication by vectors from some subset of the whole plaintext space; this is done mainly to follow Prabhakaran and Rosulek. In particular, we consider three meaningful restrictions: no restrictions, in which case we just obtain vector multiplication; the restriction to a space generated by a single value, in which case we obtain scalar multiplication; and the restriction to the identity transformation, in which case we obtain RCCA security (and in particular, because our scheme is re-randomizable, we obtain re-randomizable RCCA encryption).

Theorem 5.2. *If the encryption scheme is IND-CPA secure and the NIZK satisfies controlled-malleable simulation-sound extractability with respect to some set of transformations $\mathcal{T}$, the encryption scheme in the above construction is CM-CCA secure with respect to the same class $\mathcal{T}$.*

To prove this, we can go through the following series of game transitions, formally proved indistinguishable in Appendix F:

- Game G_0. Change to a simulated CRS and simulated proofs π in the E oracle; this is indistinguishable from $\mathsf{CM\text{-}CCA}_0$ by zero knowledge.
- Game G_1. Switch to using SimExt and $\mathsf{Extract}$ rather than Dec in the D oracle; this is indistinguishable from G_0 by the CM-SSE property of the NIZK.
- Game G_2. Switch the ciphertexts returned by the E oracle to be encryptions of random values; this is indistinguishable from G_1 by the IND-CPA security of the encryption scheme.
- Game G_3. Switch back to decrypting in the D oracle; this is now $\mathsf{CM\text{-}CCA}_1$ (and is indistinguishable from G_2 again by the CM-SSE property of the NIZK).

In addition to achieving CM-CCA security, we would also like to exploit the strong derivation privacy of the cm-NIZK to obtain a function-private encryption scheme; it turns out that if our base encryption scheme satisfies function privacy, we get this property for free.

Theorem 5.3. *If the encryption scheme is function private and the proof is strongly derivation private and zero knowledge, the encryption scheme in the above construction is function private.*

Proof. (Sketch.) To show this, we must prove that, for an adversarially chosen (c', π, T) such that $m := \mathsf{Dec}'(sk', c')$, the distribution $D_0 = (\mathsf{Enc}'(pk', T_m(m); r), \mathcal{P}(\sigma_{\mathrm{crs}}, (pk', c'), (T(m), r)))$ is indistinguishable from $D_1 = (\mathsf{Eval}'(pk', c', T), \mathsf{ZKEval}(\sigma_{\mathrm{crs}}, T, (pk', c'), \pi))$. We can demonstrate this by going through the following game transitions:

- Game G_0. Use the distribution D_0.
- Game G_1. Switch to using simulated instead of honest proofs; this is indistinguishable from G_0 by zero knowledge.
- Game G_2. Switch to using $\mathsf{Eval}'(pk', c', T)$ instead of $\mathsf{Enc}'(pk', T_m(\mathsf{Dec}'(sk', c)); r)$; this is indistinguishable from G_1 by the function privacy of the underlying encryption scheme.
- Game G_3. Switch to using $\mathsf{ZKEval}(\sigma_{\mathrm{crs}}, T, (pk', c'), \pi)$ instead of the simulator; this is now D_1 and is indistinguishable from G_2 by the strong derivation privacy of the underlying proof system.

To see how to switch from G_0 to G_1, we can take an adversary $\mathcal{A}$ that distinguishes between them and use it to construct an adversary $\mathcal{B}$ that breaks zero knowledge. To start, $\mathcal{B}$ will get a CRS σ_{crs} and can then generate (pk', sk') on its own and give to $\mathcal{A}$ $(pk', \sigma_{\text{crs}})$. When it is given (c', π, T), it can first check that $\mathcal{V}(\sigma_{\text{crs}}, (pk', c'), \pi) = 1$ (and output $\perp$ if this does not hold), then compute $m := \mathsf{Dec}'(sk', c')$, honestly compute $c'' := \mathsf{Enc}'(pk', T_m(m); r)$, query its own oracle on $((pk', c'), (T_m(m), r))$ to get back a proof π', and give (c'', π') back to $\mathcal{A}$. If $\mathcal{A}$ guesses it is in G_0 then $\mathcal{B}$ will guess it is interacting with a prover, and if $\mathcal{A}$ guesses it is in G_1 then $\mathcal{B}$ will guess it is interacting with a simulator.

To see how to switch from G_1 to G_2, we can take an adversary $\mathcal{A}$ that distinguishes between them and use it to construct an adversary $\mathcal{B}$ that breaks function privacy. To start, $\mathcal{B}$ will get a keypair (pk', sk'), generate $(\sigma_{\text{crs}}, \tau_s) \xleftarrow{\$} S_1(1^k)$, and give to $\mathcal{A}$ $(pk', \sigma_{\text{crs}})$. When it is given (c', π, T), it can give (c', T) to its own oracle to get back a ciphertext c''; it will also compute $\pi' \xleftarrow{\$} S_2(\sigma_{\text{crs}}, \tau_s, (pk', c'))$ and give (c'', π') back to $\mathcal{A}$. If $\mathcal{A}$ guesses it is in G_1 then $\mathcal{B}$ will guess $b = 0$ (i.e., it is getting back fresh encryptions) and if $\mathcal{A}$ guesses it is in G_2 then $\mathcal{B}$ will guess $b = 1$ (i.e., it is getting encryptions from Eval).

Finally, to see how to switch from G_2 to G_3, we can take an adversary $\mathcal{A}$ that distinguishes between them and use it to construct an adversary $\mathcal{B}$ that breaks strong derivation privacy; note that the behavior of $\mathcal{B}$ here is identical to the one for switching from G_0 to G_1 except for its output at the end. To start then, $\mathcal{B}$ will get a CRS σ_{crs} and can then generate (pk', sk') on its own and give to $\mathcal{A}$ $(pk', \sigma_{\text{crs}})$. When it is given (c', π, T), it can first check that $\mathcal{V}(\sigma_{\text{crs}}, (pk', c'), \pi) = 1$ (and output $\perp$ if this does not hold), then compute $m := \mathsf{Dec}'(sk', c')$, honestly compute $c'' := \mathsf{Enc}'(pk', T_m(m); r)$, query its oracle on $((pk', c'), (T_m(m), r))$ to get back a proof π', and give (c'', π') back to $\mathcal{A}$. If $\mathcal{A}$ guesses it is in G_2 then $\mathcal{B}$ will guess $b = 1$ (i.e., it is getting proofs from a simulator), and if $\mathcal{A}$ guesses it is in G_3 then $\mathcal{B}$ will guess $b = 0$ (i.e., it is getting proofs from ZKEval). □

6 Compactly Proving Correctness of a Shuffle

As described in the introduction, we achieve a notion of verifiability for shuffles that does not require each mix server to output its own proof of correctness; instead, using the malleability of our proofs, each mix server can maul the proof of the previous one rather than generate its own fresh proof. One point that it is important to keep in mind with this approach is that the soundness of the proof does not follow directly from the soundness of each of the individual proofs anymore; instead, one proof must somehow suffice to prove the validity of the entire series of shuffles, yet still remain compact. To capture this requirement, we define a new notion for the security of a shuffle, that we call *compact verifiability*.

To define our notion, we assume that a verifiable shuffle consists of three algorithms: a Setup algorithm that outputs the parameters for the shuffle and the identifying public keys for the honest mix servers, a $\mathsf{Shuffle}$ algorithm that takes in a set of ciphertexts and outputs both a shuffle of these ciphertexts and a proof that the shuffle was done properly, and finally a Verify algorithm that checks the validity of the proofs.

In our definition, the adversary is given the public keys of all the honest shuffling authorities, as well as an honestly generated public key for the encryption scheme. It can then provide a list of ciphertexts and ask that they be shuffled by one of the honest authorities (we call this an initial shuffle), or provide a set of input ciphertexts, a set of shuffled ciphertexts, and a proof, and ask one of the honest authorities to shuffle the ciphertexts again and update the proof. Finally, the adversary produces challenge values consisting of a set of input ciphertexts, a set of shuffled ciphertexts and a proof that includes the public key of at least one of the honest authorities. If this

proof verifies, it receives either the decryption of the shuffled ciphertexts, or a random permutation of the decryptions of the initial ciphertexts. Our definition requires that it should be hard for the adversary to distinguish which of the two it is given.

We also require that the input ciphertexts are always accompanied by a proof that they are well-formed; i.e., a proof of knowledge of a valid message and the randomness used in encryption. This is usually necessary in many applications (for example in voting when each voter must prove that he has encrypted a valid vote), and in our construction it means that we can easily handle an adversary who produces the input ciphertexts in invalid ways; e.g., by mauling ciphertexts from a previous shuffle, or by submitting malformed ciphertexts.

Definition 6.1. *For a verifiable shuffle* $(\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ *with respect to an encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, a given adversary* $\mathcal{A}$ *and a bit* $b \in \{0,1\}$*, let* $p_b^{\mathcal{A}}(k)$ *be the probability that* $b' = 0$ *in the following experiment:*

- *Step 1.* $(params, sk, S = \{pk_i\}, \{sk_i\}) \xleftarrow{\$} \mathsf{Setup}(1^k)$.
- *Step 2.* $\mathcal{A}$ *gets params,* S*, and access to the following two oracles: an initial shuffle oracle that, on input* $(\{c_i, \pi_i\}, pk_\ell)$ *for* $pk_\ell \in S$*, outputs* $(\{c'_i\}, \pi, \{pk_\ell\})$ *(if all the proofs of knowledge* π_i *verify), where* π *is a proof that the* $\{c'_i\}$ *constitute a valid shuffle of the* $\{c_i\}$ *performed by the user corresponding to* pk_ℓ *(i.e., the user who knows* sk_ℓ*), and a shuffle oracle that, on input* $(\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\}, pk_m)$ *for* $pk_m \in S$*, outputs* $(\{c''_i\}, \pi', \{pk_j\} \cup pk_m)$.
- *Step 3. Eventually,* $\mathcal{A}$ *outputs a tuple* $(\{c_i, \pi_i\}, \{c'_i\}, \pi, S' = \{pk_j\})$.
- *Step 4. If* $\mathsf{Verify}(params, (\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\})) = 1$ *and* $S \cap S' \neq \emptyset$ *then continue; otherwise simply abort and output* $\perp$*. If* $b = 0$ *then give* $\mathcal{A}$ $\{\mathsf{Dec}(sk, c'_i)\}$*, and if* $b = 1$ *then give* $\mathcal{A}$ $\varphi(\{\mathsf{Dec}(sk, c_i)\})$*, where* φ *is a random permutation* $\varphi \xleftarrow{\$} S_n$.
- *Step 5.* $\mathcal{A}$ *outputs a guess bit* b'.

We say that the shuffle is compactly verifiable *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

Our compactly-verifiable shuffle construction will utilize four building blocks: a *hard relation* R_{pk} (as defined by Damgård [18, Definition 3]), a re-randomizable IND-CPA-secure encryption scheme $(\mathsf{KeyGen}, \mathsf{ReRand}, \mathsf{Enc}, \mathsf{Dec})$, a proof of knowledge $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, and a cm-NIZK $(\mathsf{CRSSetup}', \mathcal{P}', \mathcal{V}')$. The hard relation will be used to ensure that the secret key sk_j known to the j-th mix server cannot be derived from its public key pk_j,[8] the proof of knowledge will be created by the users performing the initial encryptions to prove knowledge of their votes, and the cm-NIZK will be used to prove that a given collection $\{c'_i\}$ is a valid shuffle of a collection $\{c_i\}$, performed by the mix servers corresponding to a set of public keys $\{pk_j\}$. This means that the instances are of the form $x = (pk, \{c_i\}, \{c'_i\}, \{pk_j\})$, witnesses are of the form $w = (\varphi, \{r_i\}, \{sk_j\})$ (where φ is the permutation used, $\{r_i\}$ is the randomness used to re-randomize the ciphertexts, and $\{sk_j\}$ are the secret keys corresponding to $\{pk_j\}$), and the relation R is $((pk, \{c_i\}, \{c'_i\}, \{pk_j\}_{i=1}^{\ell'}), (\varphi, \{r_i\}, \{sk_j\})) \in R$ iff $\{c'_i\} = \{\mathsf{ReRand}(pk, \varphi(c_i); r_i)\} \wedge (pk_j, sk_j) \in R_{pk}$ $\forall j \in [1..\ell']$. The valid transformations are then $T_{(\varphi', \{r'_i\}, \{sk_j^+, pk_j^+\}, \{pk_j^-\})} = (T_x, T_w)$, where $T_x(pk, \{c_i\}, \{c'_i\}, \{pk_j\}) := (pk, \{c_i\}, \{\mathsf{ReRand}(pk, \varphi'(c_i); r'_i)\}, \{pk_j\} \cup (\{pk_j^+\} \setminus \{pk_j^-\}))$ and T_w transforms the witness accordingly. We combine all these primitives as follows:

[8] It is worth mentioning that generically we can use a one-way function to obtain this property, but that we cannot efficiently instantiate this in our setting and so use instead a hard relation (for more on this see Appendix D.3).

- $\mathsf{Setup}(1^k)$: To allow users to encrypt their values, run $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$ and $\sigma_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$. Next, each mix server i will run the generator for the hard relation to obtain a pair $(pk_i, sk_i) \in R_{pk}$; this will result in a collection $S := \{pk_i\}$ of public keys. Finally, run $\sigma'_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}'(1^k)$, and output $(params := (pk, \sigma_{\text{crs}}, \sigma'_{\text{crs}}), sk, S, \{sk_i\})$.

- $\mathsf{Enc}(params, \{m_i\}_{i=1}^n)$. Each user i can perform an encryption of his message m_i as $c_i \xleftarrow{\$} \mathsf{Enc}(pk, m_i)$, as well as form a proof of knowledge $\pi_i \xleftarrow{\$} \mathcal{P}(\sigma_{\text{crs}}, c_i, m_i)$ of the value contained inside this ciphertext. This produces a collection $\{(c_i, \pi_i)\}_{i=1}^n$.

- $\mathsf{Shuffle}(params, \{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\})$: First check if $\pi = \perp$ and $\{c'_i\} = \{pk_j\} = \emptyset$; if these hold, then this is the initial shuffle and we proceed as follows: first, check the validity of the proofs by running $\mathcal{V}(\sigma_{\text{crs}}, c_i, \pi_i)$ for all i, $1 \leq i \leq n$. If verification fails for some value of i abort and output $\perp$. Otherwise, perform the shuffle by picking a random permutation $\varphi \leftarrow S_n$ and computing $(c'_1, \ldots, c'_n) \xleftarrow{\$} \mathsf{ReRand}(pk, \varphi(\{c_i\}))$. Now finish by forming a proof π for the shuffle performed by the user in possession of the secret key corresponding to pk_1. Output the tuple $\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_1\})$.

  In the case that $\pi \neq \perp$ and $\{c'_i\} \neq \{pk_j\} \neq \emptyset$, we are working with some intermediate mix server, call it the k-th. We start the same as before: check the validity of the proofs π_i by running $\mathcal{V}(\sigma_{\text{crs}}, c_i, \pi_i)$ for all i, $1 \leq i \leq n$; here, we check also the validity of the proof π by running $\mathcal{V}'(\sigma'_{\text{crs}}, (pk, \{c_i\}, \{c'_i\}, \{pk_j\}), \pi)$. If any of these proofs do not pass verification abort and output $\perp$; otherwise, continue by choosing a random permutation $\varphi \xleftarrow{\$} S_n$. Next, pick randomness $\{r_i\}$ for the encryption scheme and compute $c''_i \xleftarrow{\$} \mathsf{ReRand}(pk, \varphi(c'_i); r_i)$ for all i, $1 \leq i \leq n$. Finally, define $T := T_{(\varphi, \{r_i\}, \{sk_k, pk_k\}, \emptyset)}$ and run $\mathsf{ZKEval}(\sigma'_{\text{crs}}, T, (pk, \{c_i\}, \{c'_i\}, \{pk_j\}), \pi))$ to obtain a proof π' that the values $\{c''_i\}$ constitute a valid shuffle of the $\{c_i\}$ performed by the users in possession of the secret keys corresponding to the public keys $(pk_1, \ldots, pk_k)$. Output the tuple $(\{c_i, \pi_i\}, \{c''_i\}, \pi', \{pk_j\} \cup pk_k)$.

- $\mathsf{Verify}(params, \{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\})$: Upon receiving this tuple, the verifier will first run the verification for each proof of knowledge π_i by running $\mathcal{V}(\sigma_{\text{crs}}, c_i, \pi_i)$ for all i, $1 \leq i \leq n$; if this fails for any value of i abort and return 0. Otherwise, move on to check the proof π by running $\mathcal{V}'(\sigma'_{\text{crs}}, (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi)$; again, if this fails output 0 and otherwise output 1.

Our efficient shuffle instantiation can be found in Appendix D.3.

Theorem 6.2. *If the encryption scheme is re-randomizable and IND-CPA secure, R_{pk} is a hard relation, the proofs π_i are NIZKPoKs, and the proof π is a cm-NIZK, then the above construction gives a compactly verifiable shuffle.*

To prove this theorem, we work with the following progression of games, formally proved indistinguishable in Appendix G:

- Game G_0. The honest game for $b = 0$.

- Game G_1. In Step 1 we switch to using a simulated CRS σ'_{crs}, and in Step 2 we switch to using simulated proofs π in the initiation oracle. This is indistinguishable from G_0 by zero knowledge.

- Game G_2. In Step 1 we switch to using a simulated proof π in the regular shuffle oracle as well. This is indistinguishable from G_1 by strong derivation privacy (as defined in Definition 2.5).

- Game G_3. In Step 2, we switch to having both the initiation and shuffle oracles return fresh encryptions rather than re-randomizations; this is indistinguishable from G_2 by the re-randomizability of the encryption scheme.

- Game G_4. In Step 4, we extract the permutation φ from the proof π and return $\{\mathsf{Dec}(sk, \varphi(c_i))\}$. This is indistinguishable from G_2 by the CM-SSE property of the NIZK. (While CM-SSE does not guarantee that a permutation will be extracted directly, we argue in the proof of Lemma G.4 that even in the case that a instance x' and transformation T are recovered instead of a direct witness φ, a permutation can still be recovered.)

- Game G_4. In Step 1 we switch to using a trapdoor CRS σ_{crs}, and in Step 4 we use the extraction trapdoor to extract a witness m_i from each proof π_i for all i, $1 \leq i \leq n$, and use $\varphi(m_i)$ in place of $\mathsf{Dec}(sk, \varphi(c_i))$. This is indistinguishable from G_3 by the extractability property of the π_i.

- Game G_6. In Step 2, we switch to having both the initiation and shuffle oracles return encryptions of garbage; that is, for the values c'_i, rather than compute an honest shuffle they will instead pick random values $r_1, \ldots, r_n \xleftarrow{\$} \mathcal{M}$ and use $c'_i \xleftarrow{\$} \mathsf{Enc}(pk, r_i)$. This is indistinguishable from G_5 by the IND-CPA security of the encryption scheme.

- Game G_7. In Step 4, we pick a random permutation φ' and return $\{\varphi'(m_i)\}$. This is indistinguishable from G_6 by the CM-SSE property of the NIZK and the hardness of the relation R_{pk}.

- Game G_8. In Step 2, both oracles return to permuting and freshly encrypting for the ciphertexts rather than encrypting random values. This is indistinguishable from G_7 again by the IND-CPA security of the encryption scheme.

- Game G_9. In Step 4, we return to decrypting the ciphertexts c_i instead of using the values extracted from the proofs of knowledge; that is, we return $\{\mathsf{Dec}(sk, \varphi'(c_i))\}$. This is indistinguishable from G_8 again by the extractability property of the NIZKPoKs.

- Game G_{10}. In Step 2, both oracles return to performing an honest shuffle; i.e., re-randomizing the ciphertexts rather than performing fresh encryption. This is indistinguishable from G_9 again by the re-randomizability of the encryption scheme.

- Game G_{11}. In Step 2 we switch back to using honest proofs in the regular shuffle oracle. This is indistinguishable from G_{10} again by the derivation privacy of the NIZK.

- Game G_{12}. In Step 1 we switch back to an honest CRS, and in Step 2 we switch back to honest proofs in the initiation oracle. This is now the honest game for $b = 1$, and is indistinguishable from G_{11} again by zero knowledge.

Acknowledgments

Anna Lysyanskaya was supported by NSF grants 1012060, 0964379, 0831293, and by a Sloan Foundation fellowship, and Sarah Meiklejohn was supported in part by a MURI grant administered by the Air Force Office of Scientific Research and in part by a graduate fellowship from the Charles Lee Powell Foundation.

References

[1] M. Abe. Universally verifiable mix-net with verification work indendent of the number of mix-servers. In *Proceedings of EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447. Springer, 1998.

[2] M. Abe. Mix-networks on permutation networks. In *Proceedings of Asiacrypt 1999*, pages 258–273, 1999.

[3] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *Proceedings of Crypto 2010*, volume 6223 of *LNCS*, pages 209–236, 2010.

[4] M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In *Proceedings of Crypto 2011*, volume 6841 of *LNCS*, pages 649–666, 2011.

[5] M. Abe, K. Haralambiev, and M. Ohkubo. Signing on elements in bilinear groups for modular protocol design. Cryptology ePrint Archive, Report 2010/133, 2010. `http://eprint.iacr.org/2010/133`.

[6] M. Abe and F. Hoshino. Remarks on mix-networks based on permutation networks. In *Proceedings of PKC 2001*, pages 317–324, 2001.

[7] T. Acar and L. Nguyen. Revocation for delegatable anonymous credentials. In *Proceedings of PKC 2011*, pages 423–440, 2011.

[8] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Delegatable anonymous credentials. In *Proceedings of Crypto 2009*, volume 5677 of *LNCS*, pages 108–125. Springer-Verlag, 2009.

[9] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *Proceedings of Crypto 1992*, volume 740 of *LNCS*, pages 390–420. Springer-Verlag, 1992.

[10] M. Blum, A. de Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[11] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, 2004.

[12] D. Boneh, G. Segev, and B. Waters. Targeted malleability: homomorphic encryption for restricted computations. In *Proceedings of ITCS 2012*, 2012. To appear.

[13] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Proceedings of Crypto 2011*, pages 505–524, 2011.

[14] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing chosen-ciphertext security. In *Proceedings of Crypto 2003*, pages 565–582, 2003.

[15] M. Chase and M. Kohlweiss. A domain transformations for structure-preserving signatures on group elements. Cryptology ePrint Archive, Report 2011/342, 2011. `http://eprint.iacr.org/2011/342`.

[16] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[17] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proceedings of Crypto 1998*, pages 13–25, 1998.

[18] I. Damgård. On sigma protocols. `http://www.daimi.au.dk/~ivan/Sigma.pdf`.

[19] A. de Santis, G. di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Proceedings of Crypto 2001*, volume 2139 of *LNCS*, pages 566–598. Springer-Verlag, 2001.

[20] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *Proceedings of FOCS 2010*, pages 511–520, 2010.

[21] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[22] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of STOC 1990*, pages 416–426, 1990.

[23] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of Crypto 1986*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1986.

[24] G. Fuchsbauer. Commuting signatures and verifiable encryption. In *Proceedings of Eurocrypt 2011*, pages 224–245, 2011.

[25] J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Transactions on Fundamentals of Electronic, Communications and Computer Science*, 88(1):172–188, 2005.

[26] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Proceedings of Crypto 2001*, pages 368–387, 2001.

[27] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC 2009*, pages 169–178, 2009.

[28] E. Ghadafi, N. Smart, and B. Warinschi. Groth–sahai proofs revisited. In *Proceedings of PKC 2010*, pages 177–192, 2010.

[29] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of STOC 1985*, pages 186–208, 1985.

[30] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 145–160. Springer-Verlag, 2003.

[31] J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *Proceedings of Asiacrypt 2006*, volume 4284 of *LNCS*, pages 444–459. Springer-Verlag, 2006.

[32] J. Groth and S. Lu. A non-interactive shuffle with pairing-based verifiability. In *Proceedings of Asiacrypt 2007*, volume 4833 of *LNCS*, pages 51–67. Springer-Verlag, 2007.

[33] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero-knowledge for NP. In *Proceedings of Eurocrypt 2006*, volume 4004 of *LNCS*, pages 339–358. Springer-Verlag, 2006.

[34] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Proceedings of Eurocrypt 2008*, volume 4965 of *LNCS*, pages 415–432. Springer-Verlag, 2008.

[35] A. Neff. A verifiable secret shuffle and its applications to e-voting. In *Proceedings of CCS 2001*, pages 116–125, 2001.

[36] M. Prabhakaran and M. Rosulek. Rerandomizable RCCA encryption. In *Proceedings of Crypto 2007*, volume 4622 of *LNCS*, pages 517–534. Springer-Verlag, 2007.

[37] M. Prabhakaran and M. Rosulek. Homomorphic encryption with CCA security. In *Proceedings of ICALP 2008*, pages 667–678, 2008.

[38] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceedings of FOCS 1999*, pages 543–553, 1999.

[39] K. Sako and J. Kilian. Receipt-free mix-type voting scheme. In *Proceedings of Eurocrypt 1995*, volume 921 of *LNCS*, pages 393–403. Springer-Verlag, 1995.

[40] N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Proceedings of PKC 2010*, pages 420–443, 2010.

[41] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of Eurocrypt 2010*, volume 6110 of *LNCS*, pages 24–43. Springer-Verlag, 2010.

[42] D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Proceedings of Asiacrypt 2005*, pages 273–292, 2005.

A Formal Definition and Proof of Groth-Sahai Malleability

To complement our brief discussion in Section 4.1, we present here a full characterization of Groth-Sahai proofs and the operations with respect to which they are malleable. We start with a formal definition of the types of statements GS proofs can be used to prove:

Definition A.1. *Groth-Sahai proofs allow a prover to prove knowledge of values* $X = \{x_i\}_{i\in[1..m]}$ *and* $Y = \{y_i\}_{i\in[1..n]}$ *that satisfy a bilinear equation system of the following form:*

$$\left(\bigwedge_{l=1}^{\ell} \prod_{i\in[1..m]\ j\in[1..n]} e(x_i, y_j)^{\gamma_{l,i,j}} = 1\right) \wedge \left(\bigwedge_{i\in[1..m]} x_i = a_i \vee a_i = \perp\right) \wedge \left(\bigwedge_{i\in[1..n]} y_i = b_i \vee b_i = \perp\right) .$$

The constants of the pairing-product equations are defined by the proof instance $(\Gamma_1, \ldots, \Gamma_\ell, A, B)$ *consisting of exponent matrices* $\Gamma_l = \{\gamma_{l,i,j}\}_{i\in[1..m],j\in[1..n]}, \gamma_{l,i,j} \in \mathbb{F}_p$ *for* $l \in [1..\ell]$ *and group element constants* $A = \{a_i\}_{i\in[1..m]}$, $B = \{b_i\}_{i\in[1..n]}$, *with* $a_i \in \mathbb{G}_1 \cup \{\perp\}$ *and* $b_i \in \mathbb{G}_2 \cup \{\perp\}$ *(where* $\perp$ *is used to indicate that the value of the variable is unconstrained).*

In order to accomodate these unconstrained $\perp$ values in A and B we also extend our arithmetic operations as follows: we define $\tilde{+}$ and $\tilde{\cdot}$ so that $a_1 \tilde{+} a_2 = a_1 + a_2$, $a_1 \tilde{\cdot} a_2 = a_1 \cdot a_2$ for $a_1, a_2 \neq \perp$, and $a_1 \tilde{+} \perp = \perp$, $a_1 \tilde{\cdot} \perp = \perp$ for all $a_1 \in A$ (and similarly for $\perp \tilde{+} a_2$ and $\perp \tilde{\cdot} a_2$, and for all values in B). We also want to be able to compare things with $\perp$ values, so we define $\tilde{=}$ as used in the definition; i.e., $x \tilde{=} a := (x = a \vee a = \perp)$ (and similarly for elements in B).

In the body of the paper we make several simplifications to reduce complexity: (i) we use multiplicative notation; (ii) we work with symmetric bilinear groups with a pairing $e : G \times G \to G_T$, and thus consider the case only where $X = Y$, $m = n$, and $A = B$; (iii) as a short hand notation for equations, we inline group element constants, and distinguish them by their own *font*, e.g., we write $e(x, b)e(a, y) = 1$ instead of $e(x, y')e(x', y) \wedge x' = a \wedge y' = b$; (iv) to avoid using these $\perp$ values, we assume that variables are ordered such that first $n' \leq n$ variables are constrained by values $a_1, \ldots, a_{n'}$, while the remaining variables are unconstrained. We now dispense with these simplifications and revisit our informal characterization of GS malleability in Definition 4.1, which we can express more formally as follows:

Definition A.2. *(Formal) A* valid transformation *T for a Groth-Sahai instance $(\Gamma_1, \ldots, \Gamma_\ell, A, B)$ is one that can be expressed as the composition $T_{O_1} \circ \cdots \circ T_{O_{p(k)}}$ of (a polynomial number of) the following five operations ($T_{O_1} \circ T_{O_2}$ is done component wise):*

Merge equations: $\mathsf{MergeEq}(i, j)$ *adds a new equation that is the product of equations i and j.*

$T_{\mathsf{MergeEq}(i,j)} = (T_x, T_w)$, *where* $T_x(\Gamma_1, \ldots, \Gamma_\ell, A, B) := (\Gamma_1, \ldots, \Gamma_\ell, \Gamma_{\ell+1}, A, B)$ *for* $\Gamma_{\ell+1} := \Gamma_i + \Gamma_j$, *and* $T_w(X, Y) := (X, Y)$.

Merge variables: $\mathsf{MergeVar}(x, y, z, S)$ *generates a new variable z. If x and y are both constants, z will have value xy. Otherwise it will be unconstrained (i.e., have value $\perp$). For every variable w in the set S, we add the equation $e(x, w)^{-1}e(y, w)^{-1}e(z, w) = 1$.*

Let $x := x_{i_1}$, $y := x_{i_2}$, *and* $S := \{y_{s_1}, \ldots, y_{s_{|S|}}\}$. *Then* $T_{\mathsf{MergeVar}(x,y,z,S)}(\Gamma_1, \ldots, \Gamma_\ell, A, B) = (T_x, T_w)$, *where* $T_x := (\Gamma'_1, \ldots, \Gamma'_{\ell+|S|}, ((A[1] \ldots, A[m], A[i_1] \tilde{+} A[i_2]), B))$ *for*

$$\Gamma'_l[i, j] := \begin{cases} \Gamma_l[i, j] & i \in [1..m] \wedge j \in [1..n] \\ 0 & \textit{otherwise} \end{cases} \textit{ for all } l \in [1..\ell], i \in [1..m+1],\ j \in [1..n],$$

$$\Gamma'_{\ell+l}[i, j] := \begin{cases} 1 & i = m+1 \wedge j = s_l \\ -1 & (i = i_1 \vee i = i_2) \wedge j = s_l \\ 0 & \textit{otherwise} \end{cases} \textit{ for all } l \in [1..|S|], i \in [1..m+1],\ j \in [1..n]$$

and $T_w(X, Y) := (X[1], \ldots, X[m], X[i_1] + X[i_2]), Y)$.

There is a similar transformation for $\mathbb{G}_2$, $x := y_{i_1}$, $y := y_{i_2}$, *and* $S := \{x_{s_1}, \ldots, x_{s_{|S|}}\}$.[9]

Exponentiate variable: $\mathsf{ExpVar}(x, \delta, z, S)$ *generates a new variable z. If x is a constant, $z := x^\delta$, otherwise it will be unconstrained. For every variable $w \in S$, we add the equation $e(x, w)^{-\delta}e(z, w) = 1$.*

Let $x := x_{i_e}$ *and* $S := \{y_{s_1}, \ldots, y_{s_{|S|}}\}$. *Then* $T_{\mathsf{ExpVar}(x,\delta,z,S)}(\Gamma_1, \ldots, \Gamma_\ell, A, B) = (T_x, T_w)$, *where* $T_x := (\Gamma'_1, \ldots, \Gamma'_{\ell+|S|}, ((A[1] \ldots, A[m], \delta \tilde{\cdot} A[i_e]), B))$ *for*

$$\Gamma'_l[i, j] := \begin{cases} \Gamma_l[i, j] & i \in [1..m] \wedge j \in [1..n] \\ 0 & \textit{otherwise} \end{cases} \textit{ for all } l \in [1..\ell], i \in [1..m+1],\ j \in [1..n],$$

[9] For symmetric pairings they are equivalent.

$$\Gamma'_{\ell+l}[i,j] := \begin{cases} 1 & i = m+1 \wedge j = s_l \\ -\delta & i = i_e \wedge j = s_l \\ 0 & \textit{otherwise} \end{cases} \quad \textit{for all } l \in [1..|S|], i \in [1..m+1],\ j \in [1..n]$$

and $T_w(X,Y) := ((X[1],\ldots,X[m],\delta \cdot X[i_e]),Y)$.

There is a similar transformation for $\mathbb{G}_2$, $x := y_{i_e}$, *and* $S := \{x_{s_1},\ldots,x_{s_{|S|}}\}$.[10]

Add constant equation: $\mathsf{Add}(\{a_i\},\{b_j\},\{\gamma_{ij}\})$ *takes a set of constants* a_i, b_i, *satisfying a pairing product equation* $\prod e(a_i,b_j)^{\gamma_{ij}} = 1$ *and adds these variables and the new equation to the statement.*

Let $\Gamma_c := \{\gamma_{ij}\}$, $A_c := \{a_i\}$, *and* $B_c = \{b_j\}$. *Then* $T_{\mathsf{Add}(\{a_i\},\{b_j\},\{\gamma_{ij}\})}(\Gamma_1,\ldots,\Gamma_\ell,A,B) = (T_x,T_w)$, *where* $T_x := (\Gamma'_1,\ldots,\Gamma'_{\ell+1},((A\ A_c),\binom{B}{B_c}))$ *for*

$$\Gamma'_l[i,j] := \begin{cases} \Gamma_l[i,j] & i \in [1..m] \wedge j \in [1..n] \\ 0 & \textit{otherwise} \end{cases} \quad \textit{for all } l \in [1..\ell], i \in [1..m+m_c],\ j \in [1..n+n_c],$$

$$\Gamma'_{\ell+1}[i,j] := \begin{cases} \Gamma_c[i-m,j-n] & \substack{i\in[m+1..m+m_c]\wedge \\ j\in[n+1..n+n_c]} \\ 0 & \textit{otherwise} \end{cases} \quad \textit{for all } i \in [1..m+m_c],\ j \in [1..n+n_c]$$

and $T_w(X,Y) := ((XA_c),\binom{Y}{B_c})$.

Remove equation: $\mathsf{RemoveEq}(i)$ *simply removes the* i*th equation from the list. So,* $T_{\mathsf{RemoveEq}(i)}(\Gamma_1,\ldots,\Gamma_\ell,A,B) = (T_x,T_w)$, *where* $T_x := (\Gamma_1,\ldots,\Gamma_{i-1},\Gamma_{i+1},\ldots,\Gamma_\ell,A,B)$ *and* $T_w(X,Y) := (X,Y)$.

Remove variable: $\mathsf{RemoveVar}(x)$ *removes the variable* x *from the variable set iff* x *does not appear in any of the listed equations.*

Let $x := x_{i_r}$ *and* $\Gamma_l[i_r] := \vec{0}$ *for* $l \in [1..\ell]$. *Then* $T_{\mathsf{RemoveVar}(x)}(\Gamma_1,\ldots,\Gamma_\ell,A,B) = (T_x,T_w)$, *where* $T_x := (\Gamma'_1,\ldots,\Gamma'_\ell,((A[1]\ldots,A[i_r-1],A[i_r+1],\ldots A[m]),B))$ *for*

$$\Gamma'_l[i,j] := \begin{cases} \Gamma_l[i,j] & i \in [1..i_r-1] \\ \Gamma_l[i+1,j] & i \in [i_r+1..m] \end{cases} \quad \textit{for all } l \in [1..\ell],\ i \in [1..m-1],\ j \in [1..n]$$

and $T_w(X,Y) := ((X[1],\ldots,X[i_1-1],X[i_1+1],\ldots,X[m]),Y)$.

There is a similar transformation for $\mathbb{G}_2$ *and* $x := y_{i_r}$.[11]

In Appendix A.2, we will see a proof that all of these transformations are in fact supported by Groth-Sahai proofs; i.e., GS proofs are malleable with respect to each. First, we need to recall what GS proofs look like, which we do in the next section.

[10] For symmetric pairings they are equivalent.

[11] For symmetric pairings they are equivalent.

A.1 Internals of Groth-Sahai proofs

Groth-Sahai (GS) proofs require a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ of order p. We use strict additive notation, so the neutral elements of $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are 0_1, 0_2, and 0_T respectively. Similarly, we refer to the distinguished generators as 1_1, 1_2, and 1_T. We usually omit the index when it is clear from the context. We use $P_1 \overset{e}{\bullet} P_2 = e(P_1, P_2)$ as a shorthand for the pairing, so $0_1 \overset{e}{\bullet} 0_2 = 0_T$ and $1_1 \overset{e}{\bullet} 1_2 = 1_T$. We extend this notation to vectors P_1 and P_2 of m elements to mean $P_1 \overset{e}{\bullet} P_2 := \sum_{i=1}^{m} e(P_1[i], P_2[i])$.

Let $X \in \mathbb{G}_1^m$ and $Y \in \mathbb{G}_2^n$ be the witnesses known only to the prover. In additive notation, GS proofs allow us to prove statements of the form $\bigwedge_{i=1}^{\ell} \left(X \overset{e}{\bullet} \Gamma_i Y = 0\right) \wedge X \tilde{=} A \wedge Y \tilde{=} B$; an instance of the statement can therefore be given by constants ℓ, m, n, $\Gamma_i \in \mathsf{Mat}_{m \times n}(\mathbb{F}_p)$, $A \in (\mathbb{G}_1 \cup \bot)^m$ and $B \in (\mathbb{G}_2 \cup \bot)^n$. Note that by the bilinear properties of e we have that $X \overset{e}{\bullet} \Gamma_i Y = \Gamma_i^T X \overset{e}{\bullet} Y$.

We usually omit the size constants ℓ, m, n and write

$$\mathcal{E} = \{(\Gamma_1, \ldots, \Gamma_\ell, A, B) \mid \ell, m, n \in \mathbb{N}, \Gamma_i \in \mathsf{Mat}_{m \times n}(\mathbb{F}_p), A \in (\mathbb{G}_1 \cup \bot)^m, B \in (\mathbb{G}_2 \cup \bot)^n\}$$

for the set of all pairing product equations (satisfiable or not) and $\mathcal{W} = \mathbb{G}_1^* \times \mathbb{G}_2^*$ for the set of possible witnesses. We denote the pairing product relation by $R_{\mathsf{pp}} = \{(x, w) \mid x = (\Gamma_1, \ldots, \Gamma_\ell, A, B) \in \mathcal{E}, w = (X, Y), \bigwedge_{i=1}^{\ell} \left(X \overset{e}{\bullet} \Gamma_i Y = 0\right) \wedge X \tilde{=} A \wedge Y \tilde{=} B\}$ and write L_{pp} in short for $L_{R_{\mathsf{pp}}}$.

Proving an individual equation. Following Fuchsbauer [24], we first describe the proof system for a single Γ, and with explicit commitments and randomness; commitments to witnesses X and Y can be reused for proving multiple Γ_i. A core idea of Groth-Sahai proofs is to map elements of $\mathbb{G}_1$ and $\mathbb{G}_2$ into larger commitment domains and to also define pairing product equations in those domain. For the SXDH and DLIN instantiations, the domain of commitments is described by the vector spaces $\mathbb{V}_1 = \mathbb{G}_1^{\hat{m}}$ and $\mathbb{V}_2 = \mathbb{G}_2^{\hat{m}}$ (with $\hat{m} = 2$ and $\hat{m} = 3$ respectively).

We use inclusion maps $\iota_1 : \mathbb{G}_1 \to \mathbb{V}_1$, $\iota_2 : \mathbb{G}_2 \to \mathbb{V}_2$, $\iota_T : \mathbb{G}_T \to \mathbb{V}_T$ and the bilinear mapping $E : \mathbb{V}_1 \times \mathbb{V}_2 \to \mathbb{V}_T$ as defined by Ghadafi et al. [28]:

$$E((V_{11}, \ldots V_{1\hat{m}}), (V_{21}, \ldots, V_{2\hat{m}})) := \begin{pmatrix} e(V_{11}, V_{21}) & \ldots & e(V_{11}, V_{2\hat{m}}) \\ \vdots & & \vdots \\ e(V_{1\hat{m}}, V_{21}) & \ldots & e(V_{1\hat{m}}, V_{2\hat{m}}) \end{pmatrix} .$$

For $V_1 \in \mathbb{V}_1$ and $V_2 \in \mathbb{V}_2$, we write $V_1 \overset{E}{\bullet} V_2$ as a shorthand for $E(V_1, V_2)$. We extend this notation to vectors $V_1 \in \mathbb{V}_1^m$ and $V_2 \in \mathbb{V}_2^m$ to mean $V_1 \overset{E}{\bullet} V_2 := \sum_{i=1}^{m} E(V_1[i], V_2[i])$. The mappings ι_1, ι_2, ι_T are applied to vectors element wise. When it is clear from the context whether we are referring to group elements or commitments, we omit the superscripts e and E of $\bullet$.

$\mathsf{CRSSetup}(grp)$**.** The GS parameters generation takes as input a pairing group setup grp for $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; it then picks values $U_1 \in \mathbb{V}_1^{\hat{m}}$ and $U_2 \in \mathbb{V}_2^{\hat{n}}$ such that the subspaces spanned by U_1 and U_2 overlap with those produced by $\iota_1(1_1)$ and $\iota_2(1_2)$ only in $\vec{0}$, and sets $params := (grp, U_1, U_2)$.

$\mathsf{Commit}_1(params, X, r)$. To compute commitments $C \in \mathbb{V}_1^m$ to $X \in \mathbb{G}_1^m$ using randomness $r \in \mathsf{Mat}_{m \times \hat{m}}(\mathbb{F}_p)$, compute

$$C = \iota_1(X) + rU_1.$$

Similarly, $\mathsf{Commit}_2(params, Y, s)$ computes commitments $D \in \mathbb{V}_2^n$ to $Y \in \mathbb{G}_2^n$ using randomness $s \in \mathsf{Mat}_{n \times \hat{n}}(\mathbb{F}_p)$ by computing

$$D = \iota_2(Y) + sU_2 .$$

$\mathsf{Prove}(params, (\Gamma, A, B), (X, Y), r, s, T)$. The proof $\pi = (\phi, \theta)$ uses randomness $T \in \mathsf{Mat}_{\hat{n}\times\hat{m}}(\mathbb{F}_p)$ and consists of

$$\phi := r^T \Gamma D - T^T U_2 = r^T \Gamma \iota_2(Y) + r^T \Gamma s U_2 - T^T U_2,$$
$$\theta := s^T \Gamma^T \iota_1(X) + T U_1.$$

$\mathsf{Verify}(params, (\Gamma, A, B), C, D, \pi)$. A proof $\pi = (\phi, \theta)$ is verified by checking that $C \tilde{=} \iota_1(A)$, $D \tilde{=} \iota_2(B)$, and

$$C \bullet \Gamma D = \iota_T(0) + U_1 \bullet \phi + \theta \bullet U_2. \tag{1}$$

$\mathsf{RandCom}_1(params, C, \hat{r})$. To randomize commitments $C \in \mathbb{V}_1^m$ using randomness $\hat{r} \in \mathsf{Mat}_{m\times\hat{m}}(\mathbb{F}_p)$, compute

$$C' = C + \hat{r} U_1.$$

Similarly $\mathsf{RandCom}_2(params, D, \hat{s})$ computes commitments $D' \in \mathbb{V}_2^n$ using randomness $\hat{s} \in \mathsf{Mat}_{n\times\hat{n}}(\mathbb{F}_p)$ by computing

$$D' = D + \hat{s} U_2.$$

$\mathsf{RandProof}(params, (\Gamma, A, B), (C, D), \hat{r}, \hat{s}, \hat{T})$. To create a randomized proof $\pi' = (\phi', \theta')$, let $D' := D + \hat{s} U_2$, and compute

$$\phi' := \phi + \hat{r}^T \Gamma D' - \hat{T}^T U_2 = \phi + \hat{r}^T \Gamma D + \hat{r}^T \Gamma \hat{s} U_2 - \hat{T}^T U_2,$$
$$\theta' := \theta + \hat{s}^T \Gamma^T C + \hat{T} U_1.$$

Combining proofs about the same internal commitments. To extend the previous construction, we can also define a proof system for multiple pairing product equations (i.e., one with multiple Γ values rather than just one):

$\mathsf{CRSSetup}(grp)$ is unchanged.

$\mathsf{Prove}(params, (\Gamma_1, \ldots, \Gamma_\ell, A, B), (X, Y))$. Compute commitments C to X using $\mathsf{Commit}_1(params, X, r)$ with randomness $r \in \mathsf{Mat}_{m\times\hat{m}}(\mathbb{F}_p)$, where $r[i] = \{0\}^{\hat{m}}$ if $A[i] \neq \perp$ and a random element in $\mathbb{F}_p^{\hat{m}}$ otherwise. Similarly, compute commitments D to Y using randomness s. For all i, pick randomness T_i and prove equation Γ_i using $\mathsf{Prove}(params, (\Gamma_i, A, B), (X, Y), r, s, T_i)$. Output the proof $\pi := ((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$.

$\mathsf{Verify}(params, (\Gamma, A, B), \pi)$. A proof $\pi = ((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$ is verified by checking that $C \tilde{=} \iota_1(A)$, $D \tilde{=} \iota_2(B)$, and

$$C \bullet \Gamma_i D = \iota_T(0) + U_1 \bullet \phi_i + \theta_i \bullet U_2$$

for all i, $1 \leq i \leq \ell$.

A.2 Proof of malleability

Lemma A.3. *There exists an efficient procedure* ZKEval *such that given any pairing product instance x, any valid transformation $T := (T_x, T_w)$, and any accepting Groth-Sahai proof π for x,* $\mathsf{ZKEval}(\sigma_{crs}, T, x, \pi)$ *produces an accepting proof for $T_x(x)$.*

Proof. To prove the lemma, it suffices to describe a ZKEval procedure for each of the operations in Definition A.2 and show that it transforms the GS proof correctly; this means checking that Equation 1 is still satisfied by the transformed proof.

Merge equations: $\mathsf{ZKEval}(x, \pi, T_{\mathsf{MergeEq}(i,j)})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$ and return the proof $\pi' := ((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), (\phi_i + \phi_j, \theta_i + \theta_j), C, D)$; in other words, the mauled proof for instance $(\Gamma_1, \ldots, \Gamma_\ell, \Gamma_i + \Gamma_j, A, B)$ has an additional proof pair $(\phi_{\ell+1}, \theta_{\ell+1}) := (\phi_i + \phi_j, \theta_i + \theta_j)$. We verify that Equation 1 will be satisfied for this additional proof by checking that

$$\begin{aligned} C \bullet (\Gamma_i + \Gamma_j)D &= \\ C \bullet (\Gamma_i D + \Gamma_j D) &= \\ C \bullet \Gamma_i D + C \bullet \Gamma_j D &= \\ &= \iota_T(0) + U_1 \bullet \phi_i + \theta_i \bullet U_2 + \iota_T(0) + U_1 \bullet \phi_j + \theta_j \bullet U_2 \\ &= \iota_T(0) + U_1 \bullet (\phi_i + \phi_j) + (\theta_i + \theta_j) \bullet U_2. \end{aligned}$$

Merge variables: $\mathsf{ZKEval}(x, \pi, T_{\mathsf{MergeVar}(x,y,z,S)})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$, x as x_{i_1}, y as x_{i_2}, and returns the proof $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), (\vec{0}, \vec{0}), \ldots, (\vec{0}, \vec{0}), (C[1], \ldots, C[m], C[i_1] + C[i_2]), D)$.

$\Gamma'_1, \ldots, \Gamma'_\ell$ are the same as $\Gamma_1, \ldots, \Gamma_\ell$ except for an added row of $\vec{0}$, thus $C \bullet \Gamma'_i D = C \bullet \Gamma_i D$ and the same proof elements verify for the mauled equations. For $l > \ell$,

$$\begin{aligned} C \bullet \Gamma'_l D &= C[i_1] \bullet -1D[s_l] + C[i_2] \bullet -1D[s_l] + C[m+1] \bullet D[s_l] \\ &= -C[i_1] \bullet D[s_l] - C[i_2] \bullet D[s_l] + (C[i_1] + C[i_2]) \bullet D[s_l] \\ &= 0_T, \end{aligned}$$

and so a $(\vec{0}, \vec{0})$ proof will in fact verify for these equations.

Exponentiate variables: $\mathsf{ZKEval}(x, \pi, T_{\mathsf{ExpVar}(x,\delta,z,S)})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$, x as x_{i_e} and returns the proof $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), (\vec{0}, \vec{0}), \ldots, (\vec{0}, \vec{0}), (C[1], \ldots, C[m], \delta \cdot C[i_e], D)$.

$\Gamma'_1, \ldots, \Gamma'_\ell$ are the same as $\Gamma_1, \ldots, \Gamma_\ell$ except for an added row of $\vec{0}$, thus $C \bullet \Gamma'_i D = C \bullet \Gamma_i D$ and the same proof elements verify for the mauled equations. For $l > \ell$,

$$\begin{aligned} C \bullet \Gamma'_l D &= C[i_e] \bullet -\delta D[s_l] + C[m+1] \bullet D[s_l] \\ &= (-\delta C[i_1]) \bullet D[s_l] + (\delta \cdot C[i_e]) \bullet D[s_l] \\ &= 0_T, \end{aligned}$$

and so a $(\vec{0}, \vec{0})$ proof will in fact verify for these equations.

Add constant equation: $\mathsf{ZKEval}(x, \pi, T_{\mathsf{Add}(\{a_i\},\{b_j\},\{\gamma_{ij}\})})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$. Let $\Gamma_c := \{\gamma_{ij}\}$, $A_c := \{a_i\}$, and $B_c := \{b_j\}$. ZKEval returns the proof $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), (\vec{0}, \vec{0}), (C\iota_1(A_c)), \begin{pmatrix} D \\ \iota_2(B_c) \end{pmatrix})$.

$\Gamma'_1, \ldots, \Gamma'_\ell$ are the same as $\Gamma_1, \ldots, \Gamma_\ell$ except for an added rows and columns of $\vec{0}$, thus $C \bullet \Gamma'_i D = C \bullet \Gamma_i D$ and the same proof elements verify for the mauled equations.

For $\Gamma'_{\ell+1}$, we know that $A_c \bullet \Gamma_c B_c = 0_T$, and consequently $\iota_1(A_c) \bullet \Gamma_c \, \iota_2(B_c) = \iota_T(0_T)$. Now $\Gamma'_{\ell+1}$ is Γ_c extended with rows and columns of $\vec{0}$, and $A'$and B' are equal to A_c and B_c for all elements corresponding to non zero rows and columns. Consequently $C' \bullet \Gamma'_{\ell+1} D' = \iota_T(0_T)$.

Remove equation: $\mathsf{ZKEval}(x, \pi, T_{\mathsf{RemoveEq}(i)})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$ and returns the proof $((\phi_1, \theta_1), \ldots, (\phi_{i-1}, \theta_{i-1})(\phi_{i+1}, \theta_{i+1}), \ldots, (\phi_\ell, \theta_\ell), (C, D))$.

As the checks performed on the mauled proof are a subset of the checks performed on the original proofs, we know that if the original proof verified the mauled one must as well.

Remove variable: $\mathsf{ZKEval}(x, \pi, T_{\mathsf{RemoveVar}(x)})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$, x as x_{i_r} and returns the proof $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), (C[1], \ldots, C[i_r - 1], C[i_r + 1], \ldots, C[m]), D)$.

The i_r^{th} row of all Γ_l were $\vec{0}$, thus $C' \bullet \Gamma'_i D = C \bullet \Gamma_i D$ and the same proof elements verify for the mauled equations. □

A.3 Other transformations

Remove duplicate constant: $\mathsf{RemoveConstant}(x, y)$ removes variable x that is fixed to the same constant as y.

Let $x := x_{i_1}$ and $y := x_{i_2}$, such that $A[i_1] = A[i_2] \neq \perp$. Then $T_{\mathsf{RemoveConstant}(x,y)}(\Gamma_1, \ldots, \Gamma_\ell, A, B) := (\Gamma'_1, \ldots, \Gamma'_\ell, ((A[1] \ldots, A[i_1 - 1], A[i_1 + 1], \ldots A[m]), B))$ for

$$\Gamma'_l[i,j] := \begin{cases} \Gamma_l[i,j] + \Gamma_l[i_1,j] & i \in [1..i_1 - 1] \wedge i = i_2 \\ \Gamma_l[i,j] & i \in [1..i_1 - 1] \wedge i \neq i_2 \\ \Gamma_l[i+1,j] + \Gamma_l[i_1,j] & i \in [i_r + 1..m] \wedge i + 1 = i_2 \\ \Gamma_l[i+1,j] & i \in [i_r + 1..m] \wedge i + 1 \neq i_2 \end{cases}$$

for all i, j, l, $1 \leq i < m$, $1 \leq j \leq n$, $1 \leq l \leq \ell$.

There is a similar transformation for $\mathbb{G}_2$, $x := y_{i_1}$, and $y := y_{i_2}$.[12]

$\mathsf{ZKEval}(x, \pi, T_{\mathsf{RemoveConstant}(x,y)})$ parses π as $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), C, D)$, x as x_{i_1}, y as y_{i_2} and returns the proof $((\phi_1, \theta_1), \ldots, (\phi_\ell, \theta_\ell), (C[1], \ldots, C[i_1 - 1], C[i_1 + 1], \ldots, C[m]), D)$.

Proof. Note that $A[i_1] = A[i_2]$ and the row for y in Γ'_l contain the sum of the vectors $\Gamma_l[i_1]$ and $\Gamma_l[i_2]$, for $l \in [1..\ell]$.

$$\begin{aligned} C \bullet \Gamma_l D &= \sum_{i=[1..m]} E(C[i], \sum_{j=1..n} \Gamma[i,j] D[j]) \\ &= E(A[i_1], \sum_{j=1..n} \Gamma[i_1,j] D[j]) + E(A[i_2], \sum_{j=1..n} \Gamma[i_2,j] D[j]) \\ &\quad + \sum_{i=[1..m], i \neq i_1 \wedge i \neq i_2} E(C[i], \sum_{j=1..n} \Gamma[i,j] D[j]) \\ &= E(A[i_2], \sum_{j=1..n} (\Gamma[i_1,j] + \Gamma[i_2,j]) D[j]) + \sum_{i=[1..m], i \neq i_1 \wedge i \neq i_2} E(C[i], \sum_{j=1..n} \Gamma[i,j] D[j]) \\ &= \sum_{i=[1..m-1]} E(C'[i], \sum_{j=1..n} \Gamma'[i,j] D'[j]) \\ &= C' \bullet \Gamma'_l D'. \end{aligned}$$

□

Merge Proofs: $\mathsf{MergeProofs}$ combines two instances x_1 and x_2 for independently proven statements. Let x_i have ℓ_i equations, m_i variables in $\mathbb{G}_1$, and n_i variables in $\mathbb{G}_2$, then $T_{\mathsf{MergeProofs}}((\Gamma_1, \ldots, \Gamma_{\ell_1}, A_1, B_1), (\Gamma_{\ell_1+1}, \ldots, \Gamma_{\ell_1+\ell+2}, A_2, B_2)) := (\Gamma'_1, \ldots, \Gamma'_{\ell_1+\ell_2}, ((A_1, A_2), \binom{B_1}{B_2}))$ for

$$\Gamma'_l[i,j] := \begin{cases} \Gamma_l[i,j] & i \in [1..m_1] \wedge j \in [1..n_1] \\ 0 & \text{otherwise} \end{cases}$$

[12] For symmetric pairings they are equivalent.

for all $l \in [1..\ell_1]$,$i \in [1..m_1 + m_2]$, $j \in [1..n_1 + n_2]$, and

$$\Gamma'_l[i,j] := \begin{cases} \Gamma_l[i-m_1, j-n_1] & i \in [m_1+1..m_1+m_2] \wedge j \in [n_1+1..n_1+n_2] \\ 0 & \text{otherwise} \end{cases}$$

for all $l \in [\ell_1 + 1..\ell_1 + \ell_2]$,$i \in [1..m_1 + m_2]$, $j \in [1..n_1 + n_2]$.

$\mathsf{ZKEval}(x_1, x_2, \pi_1, \pi_2, T_{\mathsf{MergeProofs}})$ parses π_1 as $((\phi_1, \theta_1), \ldots, (\phi_{\ell_1}, \theta_{\ell_2}), C_1, D_1)$, π_2 as $((\phi_{\ell_1+1}, \theta_{\ell_1+1}), \ldots, (\phi_{\ell_1+\ell_2}, \theta_{\ell_1+\ell_2}), C_2, D_2)$ and returns the proof $((\phi_1, \theta_1), \ldots, (\phi_{\ell_1+\ell_2}, \theta_{\ell_1+\ell_2}), (C_1, C_2), \binom{D_1}{D_2})$.

Proof. The individual equations are verified independently and are unaffected by the merger. □

A.4 Related work on Groth-Sahai malleability

In order to demonstrate the generality of our mauling operations described above are general, we show in this section that they generalize all previous works that use the malleability of GS proofs [20, 7, 24].

Homomorphic proofs of Dodis et al. [20]. Dodis et al. [20] consider relations $R^{\mathbf{B}}_{linear} = \{(\vec{c}, \vec{x}) \mid \mathbf{B}\vec{x} = \vec{c}\}$ where $\mathbf{B} \in \mathbb{G}_1^{M \times N}$ with instances $\vec{c} = \mathbb{G}_1^M$ and witnesses $\vec{x} \in \mathbb{F}_p^N$.

We embed their relation into a pairing product equation system with instances $x = (\Gamma_1, \ldots \Gamma_M, A, B)$. Let $\mathbf{B} = (b_{1,1}, \ldots b_{1,N}, \ldots, b_{M,N})$. Each Γ_i describes a linear equation $\sum_{j=1}^{N} b_{ij} x_j = c_i$ encoded into a pairing product equation as $\sum_{i=j}^{N} b_{ij} \bullet x_j 1_2 = c_i \bullet 1_2$. The constants of the pairing product equations are $A = (c_1, \ldots, c_M, \mathbf{B})$ and $B = (\bot, \ldots, \bot, 1_2)$.

They show the following homomorphic property for linear GS proofs: given two proofs π_1 and π_2 for instances $x_1 = (\Gamma_1, \ldots \Gamma_M, (c_1, \ldots, c_M, \mathbf{B}), B)$ and $x_2 = (\Gamma_1, \ldots \Gamma_M, (c'_1, \ldots, c'_M, \mathbf{B}), B)$ such that $\mathsf{Verify}(\sigma_{\mathrm{crs}}, x_i, \pi_i) = 1$ for $i \in [1, 2]$, the proof $(\pi_1 + \pi_2)$ obtained using componentwise addition is a valid proof for instance $(\Gamma_1, \ldots \Gamma_M, (c_1 + c'_1, \ldots, c_M + c'_M, \mathbf{B}), B)$. We check that the combination of the following mauling operations has the same property:

- use $\mathsf{MergeProofs}$ to combine the two proofs into one equation system;
- use $\mathsf{RemoveConstant}$ to remove all duplicate constants;
- use $\mathsf{MergeEq}(i, M + i)$ to 'add' the two equations about c_i, for all $i \in [1..M]$;
- use $\mathsf{MergeVar}(x_j, x'_j, x''_j)$, for all $j \in [1..N]$ and $\mathsf{MergeVar}(c_1, c'_i, x'')$ for all $i \in [1..N]$ to combine the variables and constants of the proof;
- finally we remove all intermediary equations and variables.

Homomorphic proofs of Acar and Nguyen [7]. For a fixed $M \subset \{1, \ldots, m\}$, and $N \subset \{1, \ldots, n\}$, Acar and Nguyen [7] consider mauling operations on instances (Γ_1, A_1, B_1) and (Γ_2, A_2, B_2) such that

- $A_1[i] = A_2[i] \neq \bot$, for $i \in M$ and $B_1[i] = B_2[i] \neq \bot$, for $i \in N$
- $\Gamma_1[i, j] = \Gamma_2[i, j]$, for $i \in \{1, \ldots, m\} \setminus M, j \in \{1, \ldots, n\}$
- $\Gamma_1[i, j] = \Gamma_2[i, j]$, for $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\} \setminus N$

- $\Gamma_1[i,j] = \Gamma_2[i,j] = 0$ for $i \in \{1,\ldots,m\} \setminus M, j \in \{1,\ldots,n\} \setminus N$

Acar and Nguyen implicitly use a relation R^{x_0} that is parameterized by a neutral instance of the form (Γ_0, A_0, B_0) such that $\forall_{i\in\bar{M}} A_0[i] = 0$, $\forall_{i\in\bar{N}} B_0[i] = 0$ and both $\forall_{i\in\bar{M},j\in\bar{N}}\Gamma_0[i,j] = 0$ and $\forall_{i\in M,j\in N}\Gamma_0[i,j] = 0$:

$$\begin{aligned} R^{x_0} = \{((\Gamma, A, B), (X, Y)) \mid \\ \forall_{i\in M} A[i] = A_0[i] \wedge \forall_{i\in N} B[i] = B_0[i] \wedge \forall_{i\in\bar{M}\vee j\in\bar{N}}\Gamma[i,j] = \Gamma_0[i,j] \wedge \\ X \bullet \Gamma Y = 0 \wedge X \doteq A \wedge Y \doteq B\} \end{aligned}$$

Acar and Nguyen show the following homomorphic property for GS proofs for relation R_{x_0}: given two proofs π_1 and π_2 for instances $x_1 = (\Gamma_1, A_1, B_1)$ and $x_2 = (\Gamma_2, A_2, B_2)$ such that $\mathsf{Verify}(\sigma_{\mathrm{crs}}, x_i, \pi_i) = 1$ for $i \in [1,2]$, the proof $(\pi_1 + \pi_2)$ obtained using componentwise addition is a valid proof for instance (Γ, A, B), where

$$\Gamma[i,j] = \begin{cases} \Gamma_1[i,j] + \Gamma_2[i,j] & i \in M, j \in N \\ \Gamma_1[i,j] & \text{otherwise} \end{cases}, \quad A[i] = \begin{cases} A_1[i] & i \in M \\ A_1[i] + A_2[i] & \text{otherwise} \end{cases},$$
$$B[i] = \begin{cases} B_1[i] & i \in N \\ B_1[i] + B_2[i] & \text{otherwise} \end{cases}.$$

We check that the combination of the following mauling operations has the same property:

- use $\mathsf{MergeProofs}$ to combine the two proofs into one equation system;
- use $\mathsf{MergeEq}(1,2)$ to 'add' the two equations;
- use $\mathsf{RemoveConstant}$ to remove all duplicate constants for $X[i]$, $i \in M$ and $Y[i]$, $i \in N$;
- use $\mathsf{MergeVar}_1(x_j, x'_j, x''_j)$, for all $j \in \bar{M}$ and $\mathsf{MergeVar}_2(y_j, y'_j, y'')$ for all $j \in \bar{N}$ to add the variables in $\bar{M}$ and $\bar{N}$;
- finally we remove all intermediary equations and variables.

Homomorphic proofs of Fuchsbauer [24]. Fuchsbauer shows the following homomorphic property for GS proofs and an unrestricted pairing product relation: given two proofs π_1 and π_2 for instances $x_1 = (\Gamma_1, A_1, B_1)$ and $x_2 = (\Gamma_2, A_2, B_2)$ such that $\mathsf{Verify}(\sigma_{\mathrm{crs}}, x_i, \pi_i) = 1$ for $i \in [1,2]$, the proof $(\pi_1 + \pi_2)$ obtained using componentwise addition is a valid proof for instance $(\begin{pmatrix} \Gamma_1 & 0 \\ 0 & \Gamma_2 \end{pmatrix}, (A_1, A_2), \begin{pmatrix} B_1 \\ B_2 \end{pmatrix})$. The following mauling operations have the same property:

- use $\mathsf{MergeProofs}$ to combine the two proofs into one equation system;
- use $\mathsf{MergeEq}(1,2)$ to 'add' the two equations;
- finally we remove all intermediary equations.

A.5 Derived operations

The basic operations described in Definition A.2 are sufficient to describe all known ways in which Groth-Sahai instances can be transformed. Describing a given transformation in terms of these operations, however, can be fairly complex.

To help simplify this process, we therefore present some common operations below, and show that they can be derived from the basic transformation set above.

Copying Equations $\mathsf{CopyEq}(\mathsf{eq}_k)$ creates a new equation that is identical to equation eq_k.

Derivation: Run $\mathsf{Add}(1, 1, \{1\})$ to obtain an additional equation $\mathsf{eq}^1 := e(1,1) = 1$. Run $\mathsf{MergeEq}(\mathsf{eq}_k, \mathsf{eq}^1)$ to obtain equation eq'_k, which is identical to equation k except for the additional $e(11)$ term. Now, run $\mathsf{Add}(1, 1, \{-1\})$ to obtain an additional equation $\mathsf{eq}^{-1} := e(1,1)^{-1} = 1$. Run $\mathsf{MergeEq}(\mathsf{eq}'_k, \mathsf{eq}^{-1})$ to obtain equation eq''_k, which is identical to equation eq_k. Run $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$ to remove the additional equations and variables generated.

Exponentiating Equations $\mathsf{ExpEq}(\mathsf{eq}_k, \delta)$ creates a new equation that is the same as equation eq_k except that all the exponents γ_{ij} are multiplied by δ.

Derivation: This can be derived from the $\mathsf{MergeEq}$ operation essentially by a square-and-multiply technique. Note that, if δ is negative, it is equivalent to use ($\delta \mod p$), since the group order, p is public information.

Run $\mathsf{Add}(1, 1, \{1\})$ to obtain an additional equation $\mathsf{eq}_0^{\text{aux}} := e(1,1) = 1$.

We write δ as binary as $\delta = b_1 \dots b_n$.

Now, for $i = 1, \dots, n$, do the following: If $b_i = 1$, $\mathsf{MergeEq}(\mathsf{eq}_{i-1}^{\text{aux}}, \mathsf{eq}_k)$ to obtain equation $\mathsf{eq}_i^{\text{res}}$. If $b_i = 0$, run $\mathsf{CopyEq}(\mathsf{eq}_{i-1}^{\text{aux}})$ to obtain equation $\mathsf{eq}_i^{\text{res}}$. Then run $\mathsf{MergeEq}(\mathsf{eq}_i^{\text{res}}, \mathsf{eq}_i^{\text{res}})$ to obtain equation $\mathsf{eq}_i^{\text{aux}}$. Increase i, and repeat this process.

Finally, obtain $\mathsf{eq}_n^{\text{res}}$ and run $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$ to remove the additional equations and variables generated.

Copying Variables $\mathsf{CopyVar}(x, z, S)$ generates a new variable z. If x is a constant, $z = x$, otherwise it will be unconstrained. For every variable $w \in S$, we add equation $e(x,w)^{-1}e(z,w) = 1$.

Derivation: This is essentially equivalent to $\mathsf{ExpVar}(x, z, 1, S)$.

Adding Trivial Equations $\mathsf{AddTriv}(S)$ generates a new variable 1, and the equations $e(w,1) = 1$ for all $w \in S$.

Derivation: Run $\mathsf{ExpVar}(g, -1, g^{-1}, S)$ to obtain new variable g^{-1} and equations $\{\mathsf{eq}_w := e(g,w)e(g^{-1},w) = 1\}_{w\in S}$. Run $\mathsf{MergeVar}(g, g^{-1}, 1, S)$ to obtain new variable 1 and equations $\{\mathsf{eq}'_w := e(g,w)^{-1}e(g^{-1},w)^{-1}e(1,w) = 1\}_{w\in S}$. Run $\mathsf{MergeEq}(\mathsf{eq}_w, \mathsf{eq}'_w)$ for $w \in S$ to obtain $\{e(1,w) = 1\}_{w\in S}$ as desired.

Constant to Variable $\mathsf{ConstToVar}(a, z, S)$ generates a new non-constant variable z, and adds the equation $e(a,w)^{-1}e(z,w) = 1$ for each $w \in S$.[13]

Derivation: Let x be a non-constant variable. Run $\mathsf{ExpVar}(x, -1, y, S)$ to obtain new variable y, and equation set $\{\mathsf{eq}_w := e(x,w)e(y,w) = 1\}_{w\in S}$. Run $\mathsf{MergeVar}(a, y, y', S)$ to obtain new variable y', and equation set $\{\mathsf{eq}'_w := e(ay,w)^{-1}e(y',w) = 1\}_{w\in S}$. For each $w \in S$ run $\mathsf{MergeEq}(\mathsf{eq}_w, \mathsf{eq}'_w)$ to obtain the set $\{\mathsf{eq}''_w := e(a,w)^{-1}e(x,w)e(y',w) = 1\}_{w\in S}$. Run $\mathsf{MergeVar}(x, y', z, S)$ to obtain new variable z, and equation set $\{\mathsf{eq}'''_w := e(xy',w)^{-1}e(z,w) = 1\}_{w\in S}$. For each $w \in S$ run $\mathsf{MergeEq}(\mathsf{eq}''_w, \mathsf{eq}'''_w)$ to obtain $\{e(a,w)^{-1}e(z,w) = 1\}_{w\in S}$. Run $\mathsf{RemoveEq}$ on all equations in $\{\mathsf{eq}_w, \mathsf{eq}'_w, \mathsf{eq}''_w, \mathsf{eq}'''_w\}$, and $\mathsf{RemoveVar}$ on x, y, y'.

Adding Disjunction: Given instance $x_1 \in L_{\mathsf{pp}}$, we can give transforms $\mathsf{OrL}(x_0)$ and $\mathsf{OrR}(x_0)$ such that $T_{\mathsf{OrL}(x_0)}(x_1) = \mathsf{Or}(x_0, x_1)$ and $T_{\mathsf{OrR}(x_0)}(x_1) = \mathsf{Or}(x_1, x_0)$ respectively.

[13] To make this formally derivable from the basic operations, we require that every pairing product statement have at least one non-constant variable.

W.l.o.g. we describe $\mathsf{OrL}(x_0)$. Let $x_0 = (\Gamma_1^{(0)}, \ldots, \Gamma_{\ell_0}^{(0)}, A^{(0)}, B^{(0)})$ and $x_1 = (\Gamma_1^{(1)}, \ldots, \Gamma_{\ell_1}^{(1)}, A^{(1)}, B^{(1)})$. We show that the transformation $\mathsf{OrL}(x_0)$ can be derived from our five basic mauling operations.

Derivation: For $b \in \{0, 1\}$, we use $0_b, 0'_b$ and $1_b, 1'_b$ to disambiguate between different variables constrained to 0_b and 1_b respectively.

- We start with the equations $\{(\mathsf{eq}_i^{(1)}\}_{i=1}^{\ell_1}$ about variables $X^{(1)}$ and $Y^{(1)}$.
- Use Add to introduce a new constant equation $1_1 \bullet 1_2 + 1_1 \bullet 0_2 + 1_1 \bullet 1'_2 = 0_T$.
- For $i \in [1..m_1]$, run $\mathsf{CopyVar}(X^{(1)}[i], X_1[i], \{-1'_2\} \cup \{Y^{(1)}[j]\}_{j=1}^{n_1})$ to obtain new variables $X_1[i]$ and equations $\{\mathsf{eq}_{i,j}\}_{j\in[1..n_1]}$ with $\mathsf{eq}_{i,j} = -X^{(1)}[i] \bullet Y^{(1)}[j]) + X_1[i] \bullet Y^{(1)}[j] = 0_T$ and $\{\mathsf{eq}_i^{(1,=)}\}$ with $\mathsf{eq}_i^{(1,=)} = X^{(1)}[i] \bullet 1'_2 - X_1[i] \bullet 1'_2 = 0_T$.[14]
- Use $\{\mathsf{eq}_{i,j}\}_{j\in[1..n_1]}$ and $\mathsf{MergeEq}$ to 'rename' $X^{(1)}[i]$ in $\{(\mathsf{eq}_i^{(1)}\}_{i=1}^{\ell_1}$ into $X_1[i]$. We denote the resulting equations by $\{(\mathsf{eq'}_i^{(1)}\}_{i=1}^{\ell_1}$.
- Let B' be $B^{(0)}$ with $\perp$ replaced by 1. Use $\mathsf{Add}(\vec{0}_1, B', \Gamma_i^{(0)})$, $i \in [1..\ell_0]$ and $\mathsf{RemoveConstant}$ to obtain equations $\{(\mathsf{eq}_i^{(0)}\}_{i=1}^{\ell_0}$.
- Use Add, to generate $\{\mathsf{eq}_i^{(0,=)}\}_{i=1}^{m_0}$ with $\mathsf{eq}_i^{(0),=} = A^{(0)}[i] \bullet 0_2 - 0'_1 \bullet 0_2 = 1_T$. Note that $X^{(0)}[i]$ will be $A^{(0)}[i]$ for $A^{(0)}[i] \neq \perp$.
- Use $\mathsf{ConstToVar}$ on $\{\mathsf{eq}_i^{(1,=)}\}$ and $1_1 \bullet 1_2 + 1_1 \bullet 0_2 + 1_1 \bullet 1'_2 = 0_T$. to turn $1'_2$ into v_1 to get equations $\{\mathsf{eq'}_i^{(1,=)}\}$ and $1_1 \bullet 1_2 + 1_1 \bullet 0_2 + 1_1 \bullet v_1 = 0_T$.
- Use $\mathsf{ConstToVar}$ to turn 0_2 in $\{\mathsf{eq}_i^{(0,=)}\}_{i=1}^{m_0}$ and equation $1_1 \bullet 1_2 + 1_1 \bullet 0_2 + 1_1 \bullet v_1 = 0_T$. into variable v_0 to get equations $\{\mathsf{eq'}_i^{(0,=)}\}_{i=1}^{m_0}$ and $1_1 \bullet 1_2 + 1_1 \bullet v_0 + 1_1 \bullet v_1 = 0_T$.
- Use $\mathsf{ConstToVar}$ to turn the 0_1s in $\{(\mathsf{eq}_i^{(0)}\}_{i=1}^{\ell_0}$ and $\mathsf{eq'}_i^{(0,=)}$ into variable $X_0[i]$ to get equations $\{(\mathsf{eq'}_i^{(0)}\}_{i=1}^{\ell_0}$ and $\mathsf{eq''}_i^{(0,=)}$.
- Use $\mathsf{RemoveEq}$ to remove all equations except $1_1 \bullet 1_2 + 1_1 \bullet v_0 + 1_1 \bullet v_1 = 0_T$, $\{\mathsf{eq''}_i^{(0,=)}\}_{i=1}^{m_0}$, $\{\mathsf{eq'}_i^{(1,=)}\}_{i=1}^{m_1}$,$\{(\mathsf{eq'}_i^{(0)}\}_{i=1}^{\ell_0}$, and $\{(\mathsf{eq'}_i^{(1)}\}_{i=1}^{\ell_1}$.

B Proving Disjunctions and Conjunctions

There are two main types of statement we consider proving: *disjunctions* of the form "$x_0 \in L$ or $x_1 \in L$" and *conjunctions* of the form "$x_0 \in L$ and $x_1 \in L$." In this section we consider both. We first focus on how to prove disjunctions, and then examine transformations on disjunctions. We then turn to conjuctions; at first glance, these should be easy, as we can just join the equations for x_0 and x_1 into a single equation system. The situation can become a bit more complicated, however, if the instances for x_0 and x_1 share unconstrained variables, and so we consider that case here, as well as transformations on these types of conjunctions.

[14] As a tedious technicality this requires also an ExpVar on $-1'_2$ to move the $-$ from the constant into the equation.

B.1 Disjunctions of pairing product equations

As observed by Groth [31], GS proofs also allow us to prove disjunctions of statements; i.e., a proof that either $x_0 \in L_R$ or $x_1 \in L_R$, without revealing which is true. More formally, there exist efficiently computable functions Or and Or_w, such that $\mathsf{Or}(x_0, x_1) \in L_R$ iff $x_0 \in L_R \vee x_1 \in L_R$ and whenever either $(x_0, w_0) \in R$ or $(x_1, w_1) \in R$, $(\mathsf{Or}(x_0, x_1), \mathsf{Or}_w(x_0, x_1, w_1, w_2)) \in R$.

We begin by reviewing the techniques we use to prove disjunctions. To be general, we describe the main transformation for asymmetric pairings and in additive notation.

We assume for simplicity that the two sides of the 'or' do not share unconstrained variables and can thus be described by two independent instances. Suppose we have two pairing product instances $x_0 = (\Gamma_1^{(0)}, \ldots, \Gamma_{\ell_0}^{(0)}, A^{(0)}, B^{(0)})$ and $x_1 = (\Gamma_1^{(1)}, \ldots, \Gamma_{\ell_1}^{(1)}, A^{(1)}, B^{(1)})$, and we want to prove that at least one of them is in L_{pp}. We describe a function $\mathsf{Or} : \mathcal{E} \times \mathcal{E} \to \mathcal{E}$ such that $\mathsf{Or}(x_1, x_2) \in L_{\mathsf{pp}}$ iff $x_0 \in L_{\mathsf{pp}} \vee x_1 \in L_{\mathsf{pp}}$ and a function $\mathsf{Or}_w : \mathcal{E} \times \mathcal{E} \times \mathcal{W} \times \mathcal{W} \to \mathcal{W}$ such that whenever either $(x_0, w_0) \in R_{\mathsf{pp}}$ or $(x_1, w_1) \in R_{\mathsf{pp}}$, $(\mathsf{Or}(x_0, x_1), \mathsf{Or}_w(x_0, x_1, w_0, w_1)) \in R_{\mathsf{pp}}$.

We describe the two functions together: As a first step the functions add variables v_0 and v_1 corresponding to the two sides of the disjunction. They then set up the rest of the pairing product equations so that, if v_b is not 0_2, then x_b must be true. In order to ensure that one of the original instances was true, we therefore add the following equation to show that at least one of v_0, v_1 is not 0_2:

$$1_1 \bullet 1_2 + 1_1 \bullet v_0 + 1_1 \bullet v_1 = 0_T. \tag{2}$$

(The non-degenerate property of the bilinear map then guarantees that both of the v_b variables cannot equal 0_2.)

Now we have to modify the paring product equations for each statement so that they imply that each statement is true if the appropriate $v_b \neq 0_2$, and so that if $v_b = 0_2$ it is easy to generate a satisfying assignment even without a witness for instance x_b.

To this end, we note that an equation of the form $z \bullet a = 0_T$ always has a satisfying assignment when z is unconstrained: we simply set z to 0_2. Based on this concept, we modify the set of pairing product equations as follows:

We first introduce additional variables X_0 and X_1 that will replace $X^{(0)}$ and $X^{(1)}$ in $\{\Gamma^{(0)}\}$ and $\{\Gamma^{(1)}\}$ respectively.[15] The intuition is that if we know a witness for x_0, we set X_0 and X_1 such that $X_0 = X^{(0)}$ and $X_1 := \vec{0}_1$, otherwise we do the reverse. We then add the set of equations

$$X^{(0)}[i] \bullet v_0 - X_0[i] \bullet v_0 = 0_T \text{ for all } i \in [1..m_0], \text{ and}$$
$$X^{(1)}[i] \bullet v_1 - X_1[i] \bullet v_1 = 0_T \text{ for all } i \in [1..m_1].$$

The idea is that for the equations we don't know how to prove, we will set $v_b = 0_2$ and so these equations are trivially satisfied and we can set the corresponding variables $X_0[j]$ or $X_1[j]$ to be equal to anything; in particular, we can use $X_0[j] = 0_1$ or $X_1[j] = 0_1$ (depending on what b is). For the set of equations we do know how to prove, we will have to set $v_b \neq 0_2$ and so the only way we could prove that the equation is satisfied is if we knew a witness for the original equation as well.

Finally, we modify all the original equations in $\{\Gamma^{(0)}\}$ and $\{\Gamma^{(1)}\}$ to use variables X_0 and X_1 in place of $X^{(0)}$ and $X^{(1)}$ for constrained variables.

[15] In the case where the two sides do not share unconstrained variables, the $X_b[i]$for unconstrained $X^{(b)}[i]$ are not strictly needed. We omit this optimization for the sake of generality and simplicity.

B.2 Transformations on disjunctions

Conveniently, we can show that this transformation actually preserves the malleability of the underlying proofs; i.e., we can perform the same transformations on $\mathsf{Or}(x_0, x_1)$ that we perform on instances x_0 and x_1 independently. This can be stated as the following theorem:

Theorem B.1. *For any pair of valid transformations T_1, T_2, one can derive a transformation $\mathsf{LR}(T_1, T_2)$ such that if either $x_0 \in L_R$ or $x_1 \in L_R$ then $\mathsf{LR}(T_1, T_2)(\mathsf{Or}(x_0, x_1)) = \mathsf{Or}(T_1(x_0), T_2(x_1))$.*

Proof. Suppose we are given T_0 and T_1. W.l.o.g. we will consider how to generate a transformation which, given $\mathsf{Or}(x_0, x_1)$, will produce $\mathsf{Or}(T_0(x_0), x_1)$; the second transformation can then be implemented analogously.

Let $x_0 = (\Gamma_1^{(0)}, \ldots, \Gamma_{\ell_0}^{(0)}, A^{(0)}, B^{(0)})$ and $x_1 = (\Gamma_1^{(1)}, \ldots, \Gamma_{\ell_1}^{(1)}, A^{(1)}, B^{(1)})$. $\mathsf{Or}(x_0, x_1)$ consists of equations $1_1 \bullet 1_2 + 1_1 \bullet v_0 + 1_1 \bullet v_1 = 0_T$, $\{\mathsf{eq}_i^{(0,=)}\}_{i=1}^{m_0}$, $\{\mathsf{eq}_i^{(1,=)}\}_{i=1}^{m_1}$, $\{(\mathsf{eq}'^{(0)}_i\}_{i=1}^{\ell_0}$, and $\{(\mathsf{eq}'^{(1)}_i\}_{i=1}^{\ell_1}$, where $\{\mathsf{eq}_i^{(0,=)}\}_{i=1}^{m_0}$ and $\{\mathsf{eq}_i^{(1,=)}\}_{i=1}^{m_1}$ are the equality proofs between $X^{(0)}$, X_0 and $X^{(1)}$, X_1 respectively, and $\{(\mathsf{eq}'^{(0)}_i\}_{i=1}^{\ell_0}$ and $\{(\mathsf{eq}'^{(1)}_i\}_{i=1}^{\ell_1}$ correspond to the equations in x_0 and x_1.

Given a disjunction proof constructed as described above, we consider now how to apply the transformation T_0 to the left-hand side of the proof. We will show how this can be done for each of our basic operations; as any valid transformation can be represented as a set of basic operations, this will be sufficient to show the theorem.

1. Merge Equations: $\mathsf{LR}(T_{\mathsf{MergeEq}(\mathsf{eq}_i, \mathsf{eq}_j)}, T_{\mathrm{id}})$ executes the $\mathsf{MergeEq}(\mathsf{eq}'^{(0)}_i, \mathsf{eq}'^{(0)}_j)$ operation.

2. Merge Variables: Let $x = x_{i_1}, y = x_{i_2}$, and let i_3 be the position of the new variable z as it evolves during the mauling. $\mathsf{LR}(T_{\mathsf{MergeVar}(x,y,z,S)}, T_{\mathrm{id}})$ uses $\mathsf{MergeEq}(\mathsf{eq}_{i_1}^{(0,=)}, \mathsf{eq}_{i_2}^{(0,=)})$ to merge the equality proofs of $X^{(0)}[i_1]$ and $X^{(0)}[i_2]$ to get a new equality proof $\mathsf{eq}_{i_3}^{(0,=)}$. Then it uses $\mathsf{MergeVar}(X_0[i_1], X_0[i_2], X_0[i_3], \{Y^{(0)}[j]\}_{i=1}^{m_0})$ and $\mathsf{MergeVar}(X^{(0)}[i_1], X^{(0)}[i_2], X_0[i_3], \{Y^{(0)}[j]\}_{i=1}^{m_0})$ to generate both the equations "$e(x,w)^{-1}e(y,w)^{-1}e(z,w) = 1$" for $w \in S$ that will be added to $\{(\mathsf{eq}'^{(0)}_i\}_{i=1}^{\ell_0}$ and to simplify the equality proof $\mathsf{eq}_{i_3}^{(0,=)}$ by combining $X^{(0)}[i_1]$ and $X^{(0)}[i_2]$ into $X^{(0)}[i_3]$. Finally, $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$ are used to remove unused equations and variables.

3. Exponentiate Variables: Let $x = x_{i_e}$, $S = \{y_{s_1}, \ldots y_{s_{|S|}}\}$, and let i_z be the position of the new variable z as it evolves during the mauling. $\mathsf{LR}(T_{\mathsf{ExpVar}(x,\delta,z,S)}, T_{\mathrm{id}})$ uses $\mathsf{ExpEq}(\mathsf{eq}_{i_e}^{(0,=)}, \delta)$ to maul the equality proof equation for $X^{(0)}[i_e]$ to get a new equality proof equation $\mathsf{eq}_{i_z}^{(0,=)}$.

   Then it uses $\mathsf{ExpVar}(X_0[i_e], \delta, X_0[i_z], \{Y_0[s_j]\}_{j=1}^{|S|})$ both to obtain the equations corresponding to "$e(x,w)^{-\delta}e(z,w) = 1$" and to simplify the equality proof $\mathsf{eq}_{i_z}^{(0,=)}$ to replace $X^{(0)}[i_e]^{\delta}$ with $X^{(0)}[i_e]$. Finally, $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$ are used to remove unused equations and variables.

4. Add Constant Equation: $\mathsf{LR}(T_{\mathsf{Add}(\{a_i\},\{b_j\},\{\gamma_{ij}\})}, T_{\mathrm{id}})$ executes an $\mathsf{Add}(\{a_i\}, \{b_j\}, \{\gamma_{ij}\})$ operation to extend the equations $\{(\mathsf{eq}'^{(0)}_i\}_{i=1}^{\ell_0}$ with an additional equation. The $\{a_i\}$ correspond to new $X_0[i]$ variables. In turn $\mathsf{CopyVar}(X_0[i], X^{(0)}[i], \{v_0\})$ operations extend the $\{\mathsf{eq}_i^{(0,=)}\}_{i=1}^{m_0}$ equations with additional equality proofs. This also resulting in new variable $X^{(0)}[i]$. Finally $\mathsf{ConstToVar}$ is used to hide the new $X_0[i]$ variables.

5. Remove Equation: $\mathsf{LR}(T_{\mathsf{RemoveEq}(\mathsf{eq}_i)}, T_{\mathrm{id}})$ executes the operation $\mathsf{RemoveEq}(\mathsf{eq}'^{(0)}_i)$.

6. Remove Variable: Let $x = x_{i_r}$, $\mathsf{LR}(T_{\mathsf{RemoveVar}(x)}, T_{\mathrm{id}})$ executes the operations $\mathsf{RemoveEq}(\mathsf{eq}_{i_r}^{(0,=)})$, $\mathsf{RemoveVar}(X_0[i_r])$, and $\mathsf{RemoveVar}(X^{(0)}[i_r])$. □

B.3 Conjunctions of pairing product equations

For disjunctions, we considered only the case when the two sides of the 'or' shared no unconstrained variables; here, we eliminate this requirement and consider what happens for conjunctions. So, suppose we have two pairing product instances $x_0 = (\Gamma_1^{(0)}, \ldots, \Gamma_{\ell_0}^{(0)}, A^{(0)}, B^{(0)})$ and $x_1 = (\Gamma_1^{(1)}, \ldots, \Gamma_{\ell_1}^{(1)}, A^{(1)}, B^{(1)})$. We want to prove that both of them are in L_{pp} and that they coincide on the first $\tilde{m}$ variables in $\mathbb{G}_1$ and on the first $\tilde{n}$ variables in $\mathbb{G}_2$. We write $\mathcal{S}$ for $\mathbb{N} \times \mathbb{N}$ and describe functions $\mathsf{And} : \mathcal{E} \times \mathcal{E} \times \mathcal{S} \to \mathcal{E}$ and $\mathsf{And}_w : \mathcal{W} \times \mathcal{W} \times \mathcal{S} \to \mathcal{W}$:

$\mathsf{And}(x_1, x_2; \tilde{m}, \tilde{n}) = (\Gamma_1, \ldots, \Gamma_{\ell_0+\ell_1}, A, B)$, where for $k \in [1..\ell_0]$, $i \in [1..m_0 + m_1 - \tilde{m}]$, $j \in [1..n_0 + n_1 - \tilde{n}]$

$$\Gamma_k[i,j] = \begin{cases} \Gamma_k[i,j] & i \in [1..m_0] \wedge j \in [1..n_0] \\ 0 & \text{otherwise} \end{cases}$$

and for $k \in [\ell_0 + 1..\ell_0 + \ell_1]$, $i \in [1..m_0 + m_1 - \tilde{m}]$, $j \in [1..n_0 + n_1 - \tilde{n}]$

$$\Gamma_k[i,j] = \begin{cases} \Gamma_k[i,j] & i \le \tilde{m} \wedge j \le \tilde{n} \\ \Gamma_k[i, \tilde{n} + j - n_0] & i \le \tilde{m} \wedge j > n_0 \\ \Gamma_k[\tilde{m} + i - m_0, j] & i > m_0 \wedge j \le \tilde{n} \\ \Gamma_k[\tilde{m} + i - m_0, \tilde{n} + j - n_0] & i > m_0 \wedge j > n_0 \\ 0 & \text{otherwise} \end{cases}$$

$\mathsf{And}_w(w_0, w_1; \tilde{m}, \tilde{n}) = (X, Y)$ where

$$X[i] = \begin{cases} X^{(0)}[i] & i \in [1..m_0] \\ X^{(1)}[\tilde{m} + i - m_0] & \text{otherwise} \end{cases} \text{ for } i \in [1..m_0 + m_1 - \tilde{m}]$$

$$Y[i] = \begin{cases} Y^{(0)}[i] & i \in [1..n_0] \\ Y^{(1)}[\tilde{n} + i - n_0] & \text{otherwise} \end{cases} \text{ for } i \in [1..n_0 + n_1 - \tilde{n}]$$

It follows from the construction of And and And_w that $(\mathsf{And}(x_0, x_1, \tilde{m}, \tilde{n}), \mathsf{And}_w(w_0, w_1, \tilde{m}, \tilde{n})) \in R_{\mathsf{pp}}$ iff for $w_0 = (X^{(0)}, Y^{(0)})$ and $w_1 = (X^{(1)}, Y^{(1)})$

$$(x_0, w_0) \in R_{\mathsf{pp}} \wedge (x_1, w_1) \in R_{\mathsf{pp}} \wedge \forall_{i=1}^{\tilde{m}} X^{(0)}[i] = X^{(1)}[i] \wedge \forall_{i=1}^{\tilde{n}} Y^{(0)}[i] = Y^{(1)}[i].$$

The pairing product equations described by $\mathsf{And}(x_0, x_1, \tilde{m}, \tilde{n})$ correspond to the conjunction of the equations of x_0 and x_1 plus the additional equalities. It is thus sufficient to prove $\mathsf{And}(x_0, x_1, \tilde{m}, \tilde{n})$.

B.4 Transformations on conjunctions

Before we discuss transformations on conjunctions, we need the following additional definition:

Definition B.2. *We say a valid transformation T* preserves $1...\tilde{n}$ *and* $1...\tilde{m}$ *if it can be expressed as a set of basic operations which does not include* $\mathsf{RemoveVar}(X[i])$ *for any* $i \in [1...\tilde{n}]$ *or* $\mathsf{RemoveVar}(Y[j])$ *for any* $j \in [1...\tilde{m}]$.

Note that all of our derived operations can be described by a set of basic operations which does not include running $\mathsf{RemoveVar}$ on any of the input variables, so a transformation described in terms of these derived operations that doesn't explicitly call $\mathsf{RemoveVar}$ on any of the mentioned variables will also preserve those variables.

Now we are ready to present our theorem describing transforms on conjunctions:

Theorem B.3. *For any pair of valid transformations T_0 and T_1 that preserve $1...\tilde{m}$ and $1...\tilde{n}$, we derive a transformation $\mathsf{Lift}(T_0, T_1)$ such that*

$$\mathsf{Lift}(T_0, T_1; \tilde{m}, \tilde{n})(\mathsf{And}(x_0, x_1; \tilde{m}, \tilde{n})) = \mathsf{And}(T_0(x_0), T_1(x_1); \tilde{m}, \tilde{n}).$$

Proof. Suppose we are given T_0 and T_1. W.l.o.g. we will consider how to generate a transformation which, given $\mathsf{And}(x_0, x_1; \tilde{m}, \tilde{n})$, will produce $\mathsf{And}(T_0(x_0), x_1; \tilde{m}, \tilde{n})$; the second transformation can then be implemented analogously.

Let $x_0 = (\mathsf{eq}_1^{(0)}, \ldots, \mathsf{eq}_{\ell_0}^{(0)}, A^{(0)}, B^{(0)})$ and $x_1 = (\mathsf{eq}_1^{(1)}, \ldots, \mathsf{eq}_{\ell_1}^{(1)}, A^{(1)}, B^{(1)})$. $\mathsf{And}(x_0, x_1) = (\mathsf{eq'}_1^{(0)}, \ldots, \mathsf{eq'}_{\ell_0}^{(0)}, \mathsf{eq'}_1^{(1)}, \ldots, \mathsf{eq'}_{\ell_1}^{(1)}, A', B')$.

Given a conjunction proof $\pi \leftarrow \mathcal{P}(\sigma_{\text{crs}}, \mathsf{And}(x_0, x_1; \tilde{m}, \tilde{n}), \mathsf{And}_w(w_0, w_1; \tilde{m}, \tilde{n}))$ we consider now how to apply the transformation T_0 to the left-hand side of of the proof. We will show how this can be done for each of our basic operations; as any valid transformation can be represented as a set of basic operations, this will be sufficient to show the theorem.

1. Merge Equations: $\mathsf{Lift}(T_{\mathsf{MergeEq}(\mathsf{eq}_i^{(0)}, \mathsf{eq}_j^{(0)})}, T_{\text{id}})$ executes the $\mathsf{MergeEq}(\mathsf{eq'}_i^{(0)}, \mathsf{eq'}_j^{(0)})$ operation.

2. Merge Variables: Let $x = x_{i_1}, y = x_{i_2}$, $z = x_{i_3}$ (once it's created) and $S = \{y_{s_1}, \ldots y_{s_{|S|}}\}$. Let i'_1, i'_2, i'_3, $\{s'_1, \ldots s'_{|S|}\}$ be the position of the lifted variables. This does not affect the other side of the 'and' as shared variables are excluded. $\mathsf{Lift}(T_{\mathsf{MergeVar}(x,y,z,S)}, T_{\text{id}})$ uses $\mathsf{MergeVar}(X'[i'_1], X'[i'_2], X'[i'_3], \{Y'[s'_i]\}_{i=1}^{m_0})$ to generate the equations "$e(x,w)^{-1}e(y,w)^{-1}e(z,w) = 1$" for $w \in S$ that will be added to $\{(\mathsf{eq'}_i^{(0)}\}_{i=1}^{\ell_0}$ Finally, $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$ are used to remove unused equations and variables.

3. Exponentiate Variables: Let $x = x_{i_e}$, $z = x_{i_e}$ (once it's created), $S = \{y_{s_1}, \ldots y_{s_{|S|}}\}$. Let i'_e, i'_z, $\{s'_1, \ldots s'_{|S|}\}$ be the position of the lifted variables. This does not affect the other side of the 'and' as shared variables are excluded. $\mathsf{LR}(T_{\mathsf{ExpVar}(x,\delta,z,S)}, T_{\text{id}})$ uses $\mathsf{ExpVar}(X'[i'_e], \delta, X'[i'_z], \{Y_0[s'_j]\}_{j=1}^{|S|})$ to obtain the equations corresponding to "$e(x,w)^{-\delta}e(z,w) = 1$". Finally, $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$ are used to remove unused equations and variables.

4. Add Constant Equation: $\mathsf{LR}(T_{\mathsf{Add}(\{a_i\},\{b_j\},\{\gamma_{ij}\})}, T_{\text{id}})$ executes an $\mathsf{Add}(\{a_i\}, \{b_j\}, \{\gamma_{ij}\})$ operation to extend the equations $\{(\mathsf{eq'}_i^{(0)}\}_{i=1}^{\ell_0}$ with an additional equation.

5. Remove Equation: $\mathsf{LR}(T_{\mathsf{RemoveEq}(\mathsf{eq}_i^{(0)})}, T_{\text{id}})$ executes the operation $\mathsf{RemoveEq}(\mathsf{eq'}_i^{(0)})$.

6. Remove Variable: Let $x = x_{i_r}$, and let i'_r be the position of the lifted variables. This does not affect the other side of the 'and' as shared variables are excluded. $\mathsf{Lift}(T_{\mathsf{RemoveVar}(x)}, T_{\text{id}})$ executes and $\mathsf{RemoveVar}(X[i'_r])$. □

C Proof of Efficient Controlled Malleable NIZK (Theorem 4.5)

To start, we give a slightly more fleshed-out version of our CM-friendly definition (Definition 4.3):

Definition C.1. *For a relation R and a class of transformations $\mathcal{T}$, we say $(R, \mathcal{T})$ is* CM-friendly *if the following six properties hold:*

1. *Representable statements: any instance and witness of R can be represented as a set of group elements; i.e., there are efficiently computable bijections $F_s : L_R \to G^{d_s}$ for some d_s and $F_w : W_R \to G^{d_w}$ for some d_w where $W_R := \{w \mid \exists x : (x, w) \in R\}$.*
2. *Representable transformations: any transformation in $\mathcal{T}$ can be represented as a set of group elements; i.e., there is an efficiently computable bijection $F_t : \mathcal{T} \to G^{d_t}$ for some d_t.*
3. *Provable statements: we can prove the statement $(x, w) \in R$ (using the above representation for x and w) using pairing product equations; i.e., there is a pairing product statement that is satisfied by $F_s(x)$ and $F_w(w)$ iff $(x, w) \in R$.*
4. *Provable transformations: we can prove the statement "$T_x(x') = x$ for $T \in \mathcal{T}$" (using the above representations for x and T) using a pairing product equation, i.e. there is a pairing product statement that is satisfied by $F_t(T), F_s(x), F_s(x')$ iff $T \in \mathcal{T}$ and $T_x(x') = x$.*
5. *Transformable statements: for any $T \in \mathcal{T}$, there is a valid transformation $s(T)$ that takes the statement "$(x, w) \in R$" (phrased using pairing products as above) and produces the statement "$(T_x(x), T_w(w)) \in R$."*
6. *Transformable transformations: for any $T, T' \in \mathcal{T}$ there is a valid transformation $t(T)$ that takes the statement "$T_x(x') = x$ for $T \in \mathcal{T}$" (phrased using pairing products as above) and produces the statement "$T'_x \circ T_x(x') = T'_x(x)$ for $T' \circ T \in \mathcal{T}$," and that preserves[16] the variables in x'.*

With this definition suitably formalized, we restate the theorem we would like to prove:

Theorem 4.5. *Given a derivation private NIWIPoK for pairing product statements that is malleable for the set of all valid transformations, and a structure preserving signature scheme, we can construct a cm-NIZK for any CM-friendly relation and transformation class $(R, \mathcal{T})$.*

Proof. Let $(R, \mathcal{T})$ be a CM-friendly relation and transformation class. Let $(\mathsf{Setup}_{\mathsf{pp}}, \mathcal{P}_{\mathsf{pp}}, \mathcal{V}_{\mathsf{pp}}, \mathsf{ZKEval}_{\mathsf{pp}})$ be a NIWIPoK for pairing product statements that is malleable for the set of all valid transformations, and let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerifySig})$ be a structure-preserving signature scheme.

Let R_{WI} be the relation $\{((vk, x), (w, x', T, \sigma)) \mid (x, w) \in R \vee (\mathsf{Verify}(vk, \sigma, x') = 1 \wedge x = T(x') \wedge T \in \mathcal{T})\}$ needed by our generic construction. We want to consider an embedding of this language into pairing product equations. Let PP_s be the pairing product statement specified by the provable statements requirement, let $\mathsf{PP}_{\mathsf{Ver}}$ be the pairing product equations specified by the verification algorithm of the structure preserving signature scheme, and let PP_t be the pairing product equations specified by the provable transformation requirement. Let PP have variables x, y, z, we write $\mathsf{PP}(a, b)$ for the equations in which x, y are constrained by a, b. We have that $(x, w) \in R$ iff $\mathsf{PP}_s(F_s(x), F_w(w)) = \mathit{TRUE}$, $\mathsf{Verify}(vk, \sigma, x') = 1$ iff $\mathsf{PP}_{\mathsf{Ver}}(vk, F_s(x'), \sigma) = \mathit{TRUE}$, and $x = T(x') \wedge T \in \mathcal{T}$ iff $(\mathsf{PP}_t(F_s(x), F_s(x'), F_T(T)) = \mathit{TRUE}$. $\mathsf{PP}_{\mathsf{Ver}}$ and PP_t share unconstrained variables $F_s(x')$, w.l.o.g. we assume that these are the first $\tilde{n}$ variables of $\mathsf{PP}_{\mathsf{Ver}}$ and PP_t. Let $t(T)$ and $s(x)$ be transformations on transformations respectively instances.

We note that, given the above, it will be the case that if we set

$$x_{\mathsf{pp}} = \mathsf{Or}(\mathsf{PP}_s(F_s(x)), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), \mathsf{PP}_t(x); \tilde{n}))$$

[16] We say a valid transformation T preserves a variable x if T can be described by a set of basic operations that does not include $\mathsf{RemoveVar}(x)$.

and

$$w_{\mathsf{pp}} = \mathsf{Or}_w(\mathsf{PP}_s(F_s(x)), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), \mathsf{PP}_t(x); \tilde{n}), F_w(w), \mathsf{And}_w((F_s(x'), \sigma), (F_s(x'), F_T(T), \sigma), \tilde{n})),$$

then $(x_{\mathsf{pp}}, w_{\mathsf{pp}}) \in R_{\mathsf{PP}}$ iff $((vk, x), (w, x', T, \sigma)) \in R_{\mathsf{WI}}$. Thus, we can easily implement a NIWIPoK for the relation R_{WI} as:

$\mathsf{Setup}(1^k)$**:** Run $\mathsf{Setup}_{\mathsf{pp}}(1^k)$ to get σ_{crs}.

$\mathcal{P}(\sigma_{\mathbf{crs}}, x, (w, x', T, \sigma))$**:** Let

$$x_{\mathsf{pp}} = \mathsf{Or}(\mathsf{PP}_s(F_s(x)), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), \mathsf{PP}_t(x); \tilde{n})) \text{ and}$$

$$w_{\mathsf{pp}} = \mathsf{Or}_w(\mathsf{PP}_s(F_s(x)), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), \mathsf{PP}_t(x); \tilde{n}), F_w(w), \mathsf{And}_w((F_s(x'), \sigma), (F_s(x'), F_T(T), \sigma), \tilde{n})).$$

Return the proof $\pi \leftarrow \mathcal{P}_{\mathsf{pp}}(\sigma_{\mathrm{crs}}, x_{\mathsf{pp}}, w_{\mathsf{pp}})$.

$\mathcal{V}(\sigma_{\mathbf{crs}}, x, \pi)$**:** Compute $x_{\mathsf{pp}} = \mathsf{Or}(\mathsf{PP}_s(F_s(x)), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), \mathsf{PP}_t(F_s(x)); \tilde{n}))$. Output $\mathcal{V}_{\mathsf{pp}}(\sigma_{\mathrm{crs}}, x_{\mathsf{pp}}, \pi)$.

$\mathsf{ZKEval}(T, \pi)$**:** Let $T_s = s(T)$ and $T_t = \mathsf{Lift}(\mathrm{id}, t(T))$. Compute $\mathsf{ZKEval}_{\mathsf{pp}}(\mathsf{LR}(T_s, T_t), \pi)$. □

From Theorem B.1 and B.3 we have that $\mathsf{LR}(T_s, T_t)$ is a valid transformation that transforms an instance

$$x_{\mathsf{pp}} = \mathsf{Or}(\mathsf{PP}_s(F_s(x)), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), \mathsf{PP}_t(F_s(x)); \tilde{n})))$$

into an instance

$$x'_{\mathsf{pp}} = \mathsf{Or}(s(T)(\mathsf{PP}_s(F_s(x))), \mathsf{And}(\mathsf{PP}_{\mathsf{Ver}}(vk), t(T)(\mathsf{PP}_t(F_s(x),); \tilde{n}))).$$

The properties of Or and And guarantee that $x'_{\mathsf{pp}} \in L_{\mathsf{pp}}$ iff $\exists w, x', T', \sigma : ((vk, T_x(x)), (w, x', T', \sigma)) \in R_{\mathsf{WI}}$. In other words for every $T' = (T'_x, T'_w) \in \mathcal{T}$, $\mathsf{LR}(T_s, T_t)$ for $s(T')$ and $t(T')$ realizes a function $T_{\mathsf{WI}}(T') \in \mathcal{T}_{\mathsf{WI}}$. For every $T_{\mathsf{WI}}(T') = (T_{\mathsf{WI},x}, T_{\mathsf{WI},w})$ we have that $T_{\mathsf{WI},x}(vk, x) = (vk, T'_x(x))$, and $T_{\mathsf{WI},w}(w, x', T, \sigma) = (T'_w(w), x', T' \circ T, \sigma)$.

The proof system described above is thus a derivation private NIWIPoK for R_{WI} that is malleable with respect to $\mathcal{T}_{\mathsf{WI}}$. We conclude by Theorem 3.2, 3.3 and 3.4.

D Efficient Instantiations of CM-CCA-Secure Encryption and Compactly Verifiable Shuffles

D.1 BBS encryption

For all of our efficient instantiations involving IND-CPA-secure encryption, we use the Boneh-Boyen-Shacham (BBS) encryption scheme [11], which we recall works as follows:

- $\mathsf{KeyGen}(1^k)$: Compute a bilinear group G of some prime order p with generator g and pairing $e : G \times G \to G_T$. Pick random values $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p$ and set $f := g^\alpha$ and $h := g^\beta$. Publish $pk := (p, G, G_T, g, e, f, h)$ (or just $pk := (f, h)$ if the group has been specified elsewhere) and keep $sk := (\alpha, \beta)$.

- $\mathsf{Enc}(pk, m)$: Pick random values $r, s \xleftarrow{\$} \mathbb{F}_p$ and compute $u := f^r$, $v := h^s$, and $w := g^{r+s}m$; return $c := (u, v, w)$.

- $\mathsf{Dec}(sk, c)$: Parse $c = (u, v, w)$ and $sk = (\alpha, \beta)$; then compute $m := u^{-1/\alpha} v^{-1/\beta} w$.

This scheme is multiplicatively homomorphic; to see this, note that we can simply define $\mathsf{Eval}(pk, \{c_i\}, \times) = c_1 \cdot \ldots \cdot c_n$ (i.e., the homomorphic operation on ciphertexts is also multiplication). To see that this works we consider two ciphertexts $c_1 = (f^{r_1}, h^{s_1}, g^{r_1+s_1} m_1)$ and $c_2 = (f^{r_2}, h^{s_2}, g^{r_2+s_2} m_2)$, and confirm that computing $c_1 c_2 = (f^{r_1+r_2}, h^{s_1+s_2}, g^{r_1+r_2+s_1+s_2} m_1 m_2)$ does indeed give us an encryption of $m_1 m_2$. We can similarly define an algorithm for $\mathsf{ReRand}(pk, c)$ to show that the scheme is also re-randomizable: given an encryption $c = (u := f^r, v := h^s, w := g^{r+s} m)$, we compute a new encryption $c' := (u', v', w')$ of the same message m by picking $r', s' \xleftarrow{\$} \mathbb{F}_p$ and setting $u' := u \cdot f^{r'} = f^{r+r'}$, $v' := v \cdot h^{s'} = h^{s+s'}$, and $w' := w \cdot g^{r'} g^{s'} = g^{(r+r')+(s+s')} m$. By Theorem 2.10, BBS encryption is therefore function private (as defined in Definition 2.9).

D.2 An efficient instantiation of CM-CCA-secure encryption

To consider as general a set of transformations as possible, we start with messages of the form $m := (m_1, \ldots, m_n) \in \mathbb{G}^n$ and some subspace $\mathbb{H} \subseteq \mathbb{G}^n$. We can then look at the set of transformations

$$\mathcal{T}_{\mathbb{H}} = \{(T_{(a,r')} \mid a \in \mathbb{H}\}$$

with $T_{(a,r')} = (T_x, T_w)$, $T_w(m, r) = (T_m(m), T_r(r))$, and $T_m(m_1, \ldots, m_n) = (a_1 m_1, \ldots, a_n m_n)$; in other words, transformations that perform component-wise multiplications of message vectors with vectors in $\mathbb{H}$. The corresponding transformation on ciphertexts T_x is the operation such that the value m in c is multiplied by a, and the ciphertext c is also re-randomized using r'.[17] If we use vectors of BBS encryption (outlined above), then the ciphertexts will be of the form $c = (c_1, \ldots, c_n)$ for $c_i := (u_i, v_i, w_i)$, and the randomizers r will be of the form $r = (r_1, \ldots, r_n)$ for $r_i := (f^{s_i}, h^{t_i}, g^{s_i}, g^{t_i})$, so $T_x(pk, (c_1, \ldots, c_n)) = \{(u_i \cdot f^{s'_i}, v_i \cdot h^{t'_i}, a_i w_i \cdot g^{s'_i + t'_i})\}$.

With this set of transformations in mind, we consider how to efficiently instantiate the corresponding cm-NIZK of relation $R = \{((pk, c), (m, r)) \mid c = \mathsf{Enc}'(pk, m; r)\}$; to do this we use our outline from Definition 4.3, which requires us to show that our relation and class of transformations satisfy six properties:

Representable statements: Public keys and ciphertexts consist of group elements: $pk = (f, h)$, $c = \{(u_i, v_i, w_i)\}_{i=1}^n$.

Representable transforms: We represent each transform $T_{(\{a_i, r_i\})} \in \mathcal{T}_{\mathbb{H}}$ as $\{a_i, f^{s_i}, h^{t_i}, g^{s_i}, g^{t_i}\}_{i=1}^n$.

Provable statements: To prove statements, we define the following pairing product equations with unconstrained variables $\{\mathbf{m_i}, \mathbf{g^{s_i}}, \mathbf{g^{t_i}}\}_{i=1}^n$:

$$\begin{aligned} \mathsf{eq}_{1i} &:= e(u_i, g)^{-1} e(\mathbf{g^{s_i}}, f) = 1 \ \forall i \\ \mathsf{eq}_{2i} &:= e(v_i, g)^{-1} e(\mathbf{g^{t_i}}, h) = 1 \ \forall i \\ \mathsf{eq}_{3i} &:= e(w_i, g)^{-1} e(\mathbf{m_i}, g) e(\mathbf{g^{s_i}}, g) e(\mathbf{g^{t_i}}, g) = 1 \ \forall i \end{aligned}$$

Provable transformation: To prove that for public key $pk = (f, h)$, and ciphertexts $x = \{(u_i, v_i, w_i)\}$ and $x' = \{(u'_i, v'_i, w'_i)\}$, the prover knows a transformation $T_{\{a_i, r'_i\}} \in \mathcal{T}_{\mathbb{H}}$ such that $x =$

[17] This last condition is because we would like our encryption scheme to be function private; re-randomization thus guarantees that the outcome of T will be indistinguishable from a freshly-formed ciphertext.

$T_x(x')$, we make use of the following equations in unconstrained variables $\{\mathbf{a_i}, \mathbf{g^{s'_i}}, \mathbf{g^{t'_i}}\}_{i=1}^n$:[18]

$$\begin{aligned}
\mathsf{eq}^t_{1i} &:= e(u_i, g)^{-1} e(u'_i, g) e(\mathbf{g^{s'_i}}, f) = 1 \ \forall i \\
\mathsf{eq}^t_{2i} &:= e(v_i, g)^{-1} e(v'_i, g) e(\mathbf{g^{t'_i}}, h) = 1 \ \forall i \\
\mathsf{eq}^t_{3i} &:= e(w_i, g)^{-1} e(\mathbf{a_i}, g) e(w'_i, g) e(\mathbf{g^{s'_i}}, g) e(\mathbf{g^{t'_i}}, g) = 1 \ \forall i \\
&\quad \mathsf{eq}(\mathbf{a_1}, \ldots, \mathbf{a_n})
\end{aligned}$$

where $\mathsf{eq}(\mathbf{a_1}, \ldots, \mathbf{a_n})$ is the set of equations expressing that $(a_1, \ldots, a_n) \in \mathbb{H}$.

We consider a few classes of notable transformations for specific subspaces $\mathbb{H}$:

- Vector multiplication. In this case $\mathbb{H} = G^n$ and so there is no restriction on the values a_i; this means $\mathsf{eq}(\{a_i\}) = \emptyset$.
- Scalar multiplication. In this case $\mathbb{H} = \{a\}$ for a single value a. Rather than form an additional set of proofs showing that $a_1 = \ldots = a_n = a$, we could instead modify equations eq^t_{3i} to look like $e(w_i, g)^{-1} e(\mathbf{a}, g) e(w'_i, g) e(\mathbf{g^{s'_i}}, g) e(\mathbf{g^{t'_i}}, g) = 1$ for a single value a.
- The identity transformation. This is simply a special case of the previous case, but we need to add the restriction that $a = 1$; to do this, we add the equation $e(\mathbf{a}, g) = 1$. Note that in particular this gives us an RCCA-secure encryption scheme.

Transformable statements: We transform the pairing product equations $\{\mathsf{eq}_{1i}, \mathsf{eq}_{2i}, \mathsf{eq}_{3i}\}_{i=1}^n$ for an instance $(f, h, \{(u_i, v_i, w_i)\}_{i=1}^n)$ into mauled equations for an instance $(f, h, \{(\tilde{u}_i, \tilde{v}_i, \tilde{w}_i)\}_{i=1}^n)$ with $\tilde{u}_i = u_i f^{\hat{s}_i}$, $\tilde{v}_i = v_i h^{\hat{t}_i}$, and $\tilde{w}_i = \hat{a}_i w_i g^{\hat{s}_i} g^{\hat{t}_i}$ as follows:

For all i first run $\mathsf{Add}(\mathsf{eq}_i^{(1)} := e(f^{\hat{s}_i}, g)^{-1} e(g^{\hat{s}_i}, f) = 1)$, $\mathsf{Add}(\mathsf{eq}_i^{(2)} := e(h^{\hat{t}_i}, g)^{-1} e(g^{\hat{t}_i}, h) = 1)$, and $\mathsf{Add}(\mathsf{eq}_i^{(3)} := e(\hat{a}_i g^{\hat{s}_i} g^{\hat{t}_i}, g)^{-1} e(\hat{a}_i, g) e(g^{\hat{s}_i}, g) e(g^{\hat{t}_i}, g) = 1)$. Next, multiply in these equations to the originals by using $\mathsf{MergeEq}(\mathsf{eq}_{1i}, \mathsf{eq}_i^{(1)})$, $\mathsf{MergeEq}(\mathsf{eq}_{2i}, \mathsf{eq}_i^{(2)})$, and $\mathsf{MergeEq}(\mathsf{eq}_{3i}, \mathsf{eq}_i^{(3)})$ for all i to get $\{\mathsf{eq}'_{1i}, \mathsf{eq}'_{2i}, \mathsf{eq}'_{3i}\}_{i=1}^n$

Next, use $\mathsf{MergeVar}(u_i, f^{\hat{s}_i}, \tilde{u}_i, \{g\})$ to get $\mathsf{eq}_i^{(u,f)} := e(u_i f^{\hat{s}_i}, g)^{-1} e(\tilde{u}_i, g) = 1$ and $\mathsf{MergeEq}(\mathsf{eq}_{1i}, \mathsf{eq}_i^{(u,f)})$ to combine u_i and $f^{\hat{s}_i}$ in eq_{1i}. Similarly combine the pairs $(v_i, h^{\hat{t}_i})$ into the constant $\tilde{v}_i$; $(w_i, \hat{a}_i g^{\hat{s}_i} g^{\hat{t}_i})$ into the constant $\tilde{w}_i$; $(\mathbf{g^{s_i}}, g^{\hat{s}_i})$ into the unconstrained variable $\mathbf{g^{\tilde{s}_i}}$; $(\mathbf{g^{t_i}}, g^{\hat{t}_i})$ into new unconstrained variable $\mathbf{g^{\tilde{t}_i}}$; and $(\mathbf{m_i}, \hat{a}_i)$ into new unconstrained variable $\tilde{\mathbf{m}}_\mathbf{i}$ to get equations $\{\mathsf{eq}''_{1i}, \mathsf{eq}''_{2i}, \mathsf{eq}''_{3i}\}_{i=1}^n$.

Finally, remove the now obsolete equations and variables using $\mathsf{RemoveEq}$ and $\mathsf{RemoveVar}$.

Transformable transformations: We transform the pairing product equations for proving knowledge of a transformation from instance $f, h, \{(u'_i, v'_i, w'_i)\}$ to $f, h, \{(u_i, v_i, w_i)\}$, into mauled equations for proving knowledge of a transformation from from instance $f, h, \{(u'_i, v'_i, w'_i)\}$ to $f, h, \{(u_i f^{\hat{s}_i}, v_i h^{\hat{t}_i}, a_i w_i g^{\hat{s}_i} g^{\hat{t}_i})\}_{i=1}^n$ we can use the same strategy (and the same constant equations) as for transforming statements.

We use $\mathsf{MergeVar}$ to combine the pairs $(u_i, f^{\hat{s}_i})$, $(v_i, h^{\hat{t}_i})$, and $(w_i, \hat{a}_i g^{\hat{s}_i} g^{\hat{t}_i})$ into new constants; contrary to the mauling of statements we combine the pairs $(\mathbf{g^{s'_i}}, g^{\hat{s}_i})$, $(\mathbf{g^{t'_i}}, g^{\hat{t}_i})$, and $(\mathbf{a_i}, \hat{a}_i)$ into new unconstrained variables.

[18] Note that extracting $\{a_i, g^{s'_i}, g^{t'_i}\}_{i=1}^n$ is sufficient for obtaining the whole transformation as $f^{s'_i} = u_i / u'_i$ and $h^{t'_i} = v_i / v'_i$.

Note that when removing obsolete equations and variables $\{(u'_i, v'_i, w'_i)\}$ are unaffected by RemoveVar operations.

For the three transformations considered, i.e. vector multiplication, scalar multiplication, and identity transformation on plaintexts, no additional transformations on $\mathsf{eq}(\mathbf{a_1}, \ldots, \mathbf{a_n})$ are required. For more complex restrictions on $\mathbb{H}$, eq could potentially be mauled using similar Add and MergeEq operations as above.

D.3 An efficient instantiation of our compactly verifiable shuffle

We instantiate the encryption scheme $(\mathsf{KeyGen}, \mathsf{ReRand}, \mathsf{Enc}, \mathsf{Dec})$ for the shuffle using BBS encryption. For the hard relation, we first define a generator g and a value $h := g^\beta$ for some $\beta \xleftarrow{\$} \mathbb{F}_p$; these values will be publicly available. For an individual user to compute values $(pk, sk) \in R_{pk}$, he will pick $\alpha \xleftarrow{\$} \mathbb{F}_p$ and set $pk := g^\alpha$ and $sk := h^\alpha$. We can instantiate the proof of knowledge $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ using any existing (preferably non-malleable) proof of knowledge; the main challenge then lies in instantiating the cm-NIZK $(\mathsf{CRSSetup}', \mathcal{P}', \mathcal{V}')$. To this end, we again use our outline from Section 4.2:

Representable statements: We represent a proof instance by $(pk, \{u_i, v_i, w_i\}_{i=1}^n, \{u'_i, v'_i, w'_i\}_{i=1}^n, \{pk_j\}_{i=1}^{\ell'})$.

Representable transformations: We represent each transformation $T_{(\varphi', r', \{sk_j^+, pk_j^+\}, \{pk_j^-\})} \in \mathcal{T}$ as

$$T = (\{a'_{ij}\}, (f^{s'}, h^{t'}, g^{s'+t'}), \{sk_j^+, pk_j^+\}, \{pk_j^-\}),$$

where $\{a'_{ij}\}$ is the representation of the permutation matrix corresponding to φ' using group elements 1 and g.

Provable statements: To prove that $\{u'_i, v'_i, w'_i\}_{i=1}^n$ is a permutation of $\{u_i, v_i, w_i\}_{i=1}^n$, and that the owners of public keys $\{pk_j\}_{i=1}^{\ell'}$) were involved in the mixing we use the following equations with unconstrained variables $\{\mathbf{a_{ij}}\}_{1 \le i,j \le n}, \{\mathbf{g^{s_i}}, \mathbf{g^{t_i}}\}_{i=1}^n, \{\mathbf{sk_j}\}_{i=1}^{\ell'}$:

$$e(\mathbf{a_{ij}}, \mathbf{a_{ij}}/g) = 1 \qquad \prod_{k=1}^{n} e(\mathbf{a_{ik}}/g, g) = 1 \qquad \prod_{k=1}^{n} e(\mathbf{a_{kj}}/g, g) = 1 \ \forall i, j$$

$$\mathsf{eq}_i^{(u)} := e(u'_i, g)^{-1} \prod_{j=1}^{n} e(u_i, \mathbf{a_{ij}}) e(f, \mathbf{g^{s_i}}) = 1 \ \forall i$$

$$\mathsf{eq}_i^{(v)} := e(v'_i, g)^{-1} \prod_{j=1}^{n} e(v_i, \mathbf{a_{ij}}) e(h, \mathbf{g^{t_i}}) = 1 \ \forall i$$

$$\mathsf{eq}_i^{(w)} := e(w'_i, g)^{-1} \prod_{j=1}^{n} e(w_i, \mathbf{a_{ij}}) e(\mathbf{g^{s_i}}\, \mathbf{g^{t_i}}, g) = 1 \ \forall i$$

$$e(pk_j, h)^{-1} \cdot e(\mathbf{sk_j}, g) = 1 \ \forall 1 \le j \le \ell'$$

Provable transformations: To prove that for ciphertexts and public keys $x = \{(u_i, v_i, w_i)\}, \{pk_j\}$ and $x' = \{(u'_i, v'_i, w'_i), \{pk'_j\}\}$, the prover knows a transformation $T_{(\varphi, r, \{sk_j^+, pk_j^+\}, \{pk_j^-\})} \in \mathcal{T}$ such that $x = T_x(x')$, we make use of the following equations in unconstrained variables

$\{\mathbf{a_{ij}}\}_{1\le i,j\le n}, \{\mathbf{g^{s_i}}, \mathbf{g^{t_i}}\}_{i=1}^n, \{sk_{\mathbf{j}}^+\}$:

$$e(\mathbf{a_{ij}}, \mathbf{a_{ij}}/g) = 1 \qquad \prod_{k=1}^{n} e(\mathbf{a_{ik}}/g, g) = 1 \qquad \prod_{k=1}^{n} e(\mathbf{a_{kj}}/g, g) = 1 \ \forall i, j$$

$$\mathsf{eq}_i^{(u,T)} := e(u_i', g)^{-1} \prod_{j=1}^{n} e(u_i, \mathbf{a_{ij}}) e(f, \mathbf{g^{s_i}}) = 1 \ \forall i$$

$$\mathsf{eq}_i^{(v,T)} := e(v_i', g)^{-1} \prod_{j=1}^{n} e(v_i, \mathbf{a_{ij}}) e(h, \mathbf{g^{t_i}}) = 1 \ \forall i$$

$$\mathsf{eq}_i^{(w,T)} := e(w_i', g)^{-1} \prod_{j=1}^{n} e(w_i, \mathbf{a_{ij}}) e(\mathbf{g^{s_i}}\, \mathbf{g^{t_i}}, g) = 1 \ \forall i$$

$$(e(pk_j, g)^{-1} e(pk_j', g) = 1 \vee e(pk_j, h)^{-1} \cdot e(\mathbf{sk_j^+}, g) = 1 \ \forall 1 \le j \le \ell'$$

Transformable statements: Let $A = \{a_{ij}\}$ be the permutation matrix in the witness of the proof. Let $\hat{A}$ be the permutation matrix corresponding to $\hat{\varphi}$ of $T_{(\hat{\varphi},\hat{r},\{sk_j^+,pk_j^+\},\{pk_j^-\})}$; note that $(A\hat{A}) = \hat{\varphi}(\{A_{\cdot j}\}) = \hat{\varphi}(\{A_{i\cdot}\})$. This means that the shuffled permutation matrix can be obtained by shuffling either the rows or the columns of A with φ'.

As our main mauling operation we permute the variables $\{u_i', v_i', w_i'\}_{i=1}^n$, $\{\tilde{a}_{i\cdot}\}_{1\le i\le n}$ by $\hat{\varphi}$. After this transformation all equations verify, as the only thing that was changed was the order in which the same variables are paired with each other.

The second mauling operation multiplies the same additional randomness into equations $\{\mathsf{eq}_i^{(u,x)}, \mathsf{eq}_i^{(v,x)}, \mathsf{eq}_i^{(w,x)}\}_{i=1}^n$.

The last mauling operation adds equations $e(pk_j^+), h)^{-1} \cdot e(\mathbf{sk_j^+}, g) = 1$ for all $\{sk_j^+, pk_j^+\}$ and removes all equations for $\{pk_j^-\}$.

Transformable transformations: We permute the variables $\{u_i', v_i', w_i'\}_{i=1}^n$, $\{\tilde{a}_{i\cdot}\}_{1\le i\le n}$ by φ'. We multiply the same additional randomness into equations $\{\mathsf{eq}_i^{(u,T)}, \mathsf{eq}_i^{(v,T)}, \mathsf{eq}_i^{(w,T)}\}_{i=1}^n$..

We add equations $e(pk_j^+, g)^{-1} e(pk_j', g) = 1 \vee e(pk_j^+, h)^{-1} \cdot e(sk_{\mathbf{j}}^+, g) = 1$ for all $\{sk_j^+, pk_j^+\}$ and remove all equations for $\{pk_j^-\}$.

E Relating CM-CCA Security to Other Notions

In Section 5 we put forth our new notion (Definition 5.1) of controlled malleability for encryption, and showed that it can be realized by combining IND-CPA-secure encryption and a cm-NIZK. To motivate this new definition, we show that it is in fact closely related to existing notions of security. In particular, we start by showing that our definition generalizes the notions of RCCA security [14] and CCA security; we then show that it is also closely related to HCCA security [37], and additionally implies the notion of targeted malleability introduced by Boneh et al. [12].

Theorem E.1. *For the cases when $\mathcal{T} = \emptyset$ and $T = \{id\}$, CM-CCA security (as defined in Definition 5.1) implies CCA and RCCA security respectively.*

Proof. To show this, we take an adversary $\mathcal{A}$ that breaks the RCCA or CCA security of the system with some non-negligible advantage ϵ and use it to construct a $\mathcal{B}$ that breaks the CM-CCA-security

of the system with some similar non-negligible advantage ϵ'. To start, $\mathcal{B}$ gets in some public key pk; it then gives this directly to $\mathcal{A}$. On Dec queries, $\mathcal{B}$ will query its own oracle D and return the answer it obtains here. At some point, $\mathcal{B}$ will get a challenge (m_0, m_1); it can now pick its own bit b and query E on m_b to get back a ciphertext c that it can then pass along to $\mathcal{A}$. On second-stage Dec queries, it can again query its D oracle; this time, its answer will depend on whether $\mathcal{A}$ is the RCCA or CCA adversary. For the RCCA adversary, if the answer it gets back from D is m_0 or m_1 then it will send replay to $\mathcal{A}$ and otherwise it will send the answer directly; for the CCA adversary, it will always send the answer back directly unless the queried ciphertext is identical to the challenge ciphertext c (in which case it will return $\perp$). At the end, if $\mathcal{A}$ correctly guesses b then $\mathcal{B}$ will guess it is in the real world, and otherwise it will guess it is in the simulated world.

To see that interactions with $\mathcal{B}$ will be indistinguishable from those in the honest setting, we can consider all stages of the interaction. In the first stage, we know that the values $\mathcal{B}$ queries to D cannot possibly be derived from any SimEnc ciphertexts (in either the real or simulated world, as $\mathcal{B}$ has at this point made no queries to the E oracle), so in particular the value returned will be (by definition of both Dec and SimDec) the honest decryption of the queried value. In the challenge phase, we know the value given to $\mathcal{A}$ will either be an honest encryption of the chosen message m_b (in the real world) or an encryption of an entirely random message (in the simulated world); these will be indistinguishable to $\mathcal{A}$ by the IND-CPA security of the encryption scheme. In the second-stage decryption queries, we can define SimExt as follows: for a CCA adversary it will check if the input c is in Q_c; if so it outputs id and otherwise $\perp$. For an RCCA adversary, it will decrypt the ciphertext and check if the plaintext is in Q_m; if it is then it outputs id and otherwise $\perp$. Because $\mathcal{B}$ only ever makes one query to SimEnc, we know that the only ciphertext in Q_c will be the challenge ciphertext, and the only message in Q_m will be m_b; in either case then, this SimExt behavior guarantees that SimDec will return the queried message m_b (rather than the random message encrypted by SimEnc), and thus the overall behavior of $\mathcal{B}$ will be indistinguishable in this stage as well.

Finally, we see that in the simulated world, the challenge ciphertext given to $\mathcal{A}$ contains no information whatsoever about the bit b, and so in particular $\mathcal{A}$ can have no advantage here. By guessing that it is in the simulated world only when $\mathcal{A}$ correctly guesses the bit, $\mathcal{B}$ will therefore suceed with the same non-negligible advantage ϵ as $\mathcal{A}$. □

E.1 Comparison with HCCA

Our definition for CM-CCA-secure encryption is very similar to (and in fact heavily inspired by) the definition of homomorphic-CCA-secure (HCCA) encryption introduced by Prabhakaran and Rosulek [37]. To examine the similarities, we first recall the HCCA security definition here:

Definition E.2. [37] *For an encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, a set of transformations* $\mathcal{T}$*, a given adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$.
- *Step 2.* $m^* \xleftarrow{\$} \mathcal{A}^{\mathsf{GRigEnc}_{pk}(\cdot), \mathsf{GRigExtract}_{sk}(\cdot,\cdot), \mathsf{Dec}_{sk}(\cdot)}(pk)$*, where* $\mathsf{GRigEnc}_{pk}(\cdot)$ *calls* $\mathsf{RigEnc}(pk)$ *to obtain* (c_i, S_i) *(when called for the* i*-th time) and returns* c_i *and* $\mathsf{GRigExtract}_{sk}(c, i)$ *returns* $\mathsf{RigExtract}_{sk}(c, S_i)$.
- *Step 3. If* $b = 0$*, set* $c^* \xleftarrow{\$} \mathsf{Enc}(pk, m^*)$*; otherwise, call* $\mathsf{RigEnc}(pk)$ *to obtain* (c^*, S^*).

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{GRigEnc}_{pk}(\cdot),\mathsf{GRigExtract}_{sk}(\cdot,\cdot),\mathsf{RigDec}_{sk}(\cdot)}(c^*)$*, where* GRigEnc *and* GRigExtract *are the same as in Step* 2 *and* RigDec *is defined as:*

$$\mathsf{RigDec}(sk, c) = \begin{cases} T(m^*) & \text{if } \perp \neq T \leftarrow \mathsf{RigExtract}_{sk}(c, S^*) \\ \mathsf{Dec}_{sk}(c) & \text{otherwise.} \end{cases}$$

We say that the encryption scheme is homomorphic-CCA secure *(HCCA-secure for short) if there exist PPT algorithms* RigEnc *and* RigExtract *used as above such that for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.*

Right away, we can see that their RigDec algorithm corresponds closely to our SimDec algorithm, while their RigEnc and RigExtract are essentially identical to our SimEnc and SimExt respectively. Aside from these notational differences, the main difference between the definitions is that while they consider an adversary that sees only one challenge message that is either real or simulated, we consider instead an adversary that is given an encryption oracle that will output either many real encryptions or many simulated encryptions. In order to ensure that their definition will still imply multi-message security, they give the adversary oracle access to the GRigExtract and GRigEnc oracles. This allows them to describe a series of hybrid games in which one by one real encryptions are replaced by simulated encryptions, and thus argue that this 1-message definition does in fact give multi-message security.

As we achieve multi-message security by definition, we do not require direct SimExt oracle access; the fact that the adversary is given direct access to this oracle in the HCCA game means that their definition is strictly stronger than ours, and in fact our construction does not seem to satisfy their definition.[19] Our definition is implied by theirs, however, and we believe that our definition does capture all the *desired* security goals.

As evidence, we consider the UC functionality for homomorphic message posting $\mathcal{F}_{HMP}^{\mathcal{T}}$ that Prabhakaran and Rosulek propose as "a natural security definition encompassing both unlinkability and our desired notion of non-malleability." They show that $\mathcal{F}_{HMP}^{\mathcal{T}}$ can be realized by any encryption scheme that is both HCCA and unlinkable (i.e., function private); we argue that in fact it can also be realized if the scheme is only CM-CCA secure rather than HCCA (it still needs to be unlinkable). The proof follows exactly the same structure as that of Prabhakaran and Rosulek. The only difference is that in the proof of their Claim 3, (which argues that the environment can't tell whether the message handles are derived as encryptions of the (transformed) messages or as simulated encryption), we can directly apply the CM-CCA property, without any additional hybrids.

E.2 Comparison with targeted malleability

To begin the comparison between CM-CCA security and the notion of targeted malleability introduced by Boneh et al. [12], we must first recall their definition:

Definition E.3. [12] *Let $t = t(k)$ be a polynomial. A public-key encryption scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is* t-bounded non-malleable against chosen-plaintext attacks with respect to a set of functions $\mathcal{F}$ *if for any polynomials $r = r(k)$ and $q = q(k)$ and for any PPT algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a PPT algorithm $S = (S_1, S_2)$ such that the distributions $\{\mathsf{Real}^{\mathsf{CPA}}_{\Pi,\mathcal{A},t,r,q}(k)\}_{k\in\mathbb{N}}$ and $\{\mathsf{Sim}^{\mathsf{CPA}}_{\Pi,S,t,r,q}(k)\}_{k\in\mathbb{N}}$ are computationally indistinguishable, where these distributions are defined as follows:*

[19] To see this note that if the adversary is given the extraction trapdoor for the cm-NIZK, then we can no longer argue that real and simulated are indistinguishable, so we cannot argue that it is hard to distinguish real and simulated encryptions.

$\mathsf{Real}^{\mathsf{CPA}}_{\Pi,\mathcal{A},t,r,q}(k)$

1. $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$
2. $(\mathcal{M}, \mathsf{state}_1, \mathsf{state}_2) \xleftarrow{\$} \mathcal{A}_1(1^k, pk)$
3. $(m_1, \ldots, m_r) \xleftarrow{\$} \mathcal{M}$
4. $c_i^* \xleftarrow{\$} \mathsf{Enc}(pk, m_i)$ *for all* i, $1 \le i \le r$
5. $(c_1, \ldots, c_q) \xleftarrow{\$} \mathcal{A}_2(1^k, c_1^*, \ldots, c_r^*, \mathsf{state}_2)$
6. *For every* j, $1 \le j \le q$, *let*

$$d_j = \begin{cases} \mathsf{copy}_i & \text{if } c_j = c_i^* \\ \mathsf{Dec}(sk, c_j) & \text{otherwise} \end{cases}$$

7. *Output* $(\mathsf{state}_1, m_1, \ldots, m_r, d_1, \ldots, d_q)$

$\mathsf{Sim}^{\mathsf{CPA}}_{\Pi,S,t,r,q}(k)$

1. $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$
2. $(\mathcal{M}, \mathsf{state}_1, \mathsf{state}_2) \xleftarrow{\$} S_1(1^k, pk)$
3. $(m_1, \ldots, m_r) \xleftarrow{\$} \mathcal{M}$
4. $(c_1, \ldots, c_q) \xleftarrow{\$} S_2(1^k, \mathsf{state}_2)$
5. *For every* j, $1 \le j \le q$, *let*

$$d_j = \begin{cases} \mathsf{copy}_i & \text{if } c_j = \mathsf{copy}_i \\ f(m_i) & \text{if } c_j = (i, f_1, \ldots, f_\ell), 1 \le i \le r, \\ & \ell \le t, f_i \in \mathcal{F}\ \forall i, f = f_1 \circ \cdots \circ f_\ell \\ \mathsf{Dec}(sk, c_j) & \text{otherwise} \end{cases}$$

6. *Output* $(\mathsf{state}_1, m_1, \ldots, m_r, d_1, \ldots, d_q)$

As mentioned by Boneh et al., this definition can be extended to capture CCA1 attacks by allowing $\mathcal{A}_1$ to have oracle access to $\mathsf{Dec}_{sk}(\cdot)$ in Step 2 (i.e., when choosing the distribution $\mathcal{M}$), and leaving the simulated distribution the same.

One of the main contributions of this definition is that it allows for function classes that are not necessarily closed under composition; as we can see, this is done by extracting the entire list of functions $(f_1, \ldots, f_\ell)$ and requiring that each one be in the class $\mathcal{F}$, rather than just their composition f. In our model, this can be expressed by defining the class of transformations $\mathcal{T}$ to be transformations of the form $(f_1, \ldots, f_n)$ for $n < t$; similarly, $T \circ T'$ can be represented as $(f_1, \ldots, f_n, f'_1, \ldots, f'_n)$, where $T(x) = f_1 \circ f_2 \circ \cdots \circ f_n(x)$. If we do this, then we can see that our definition of CM-CCA security in fact implies targeted malleability (with the caveat that in representing our transformations this way the size of our proofs must necessarily grow).

Theorem E.4. *For any $\mathcal{F}$, if $\mathcal{T}$ is derived from $\mathcal{F}$ as above, then any encryption scheme that is CM-CCA secure will also be t-bounded non-malleable against CCA1 attacks with respect to $\mathcal{F}$.*

Proof. To show this, we must take an adversary $\mathcal{D}$ that distinguishes between the real distribution for some adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and the simulated distribution for a simulator (S_1, S_2) with some non-negligible advantage ϵ, and use both $\mathcal{D}$ and $(\mathcal{A}_1, \mathcal{A}_2)$ to construct an adversary $\mathcal{B}$ that can win at the CM-CCA game with related non-negligible advantage ϵ'.

To start, we describe our simulator (S_1, S_2) based on $(\mathcal{A}_1, \mathcal{A}_2)$. Let $\mathsf{SimKeyGen}, \mathsf{SimEnc}, \mathsf{SimExt}$ be as specified in Section 5. The simulator $S_1(pk)$ will first run $(pk', sk', \tau_1, \tau_2) \xleftarrow{\$} \mathsf{SimKeyGen}(1^k)$. It will then run $\mathcal{A}_1(pk')$ and output the same $(\mathcal{M}, \mathsf{state}_1, \mathsf{state}_2)$ as it does. If $\mathcal{A}_1$ makes any decryption queries, S_1 can answer them honestly using the secret key sk'. Given state_2, S_2 can run $c_i^* \xleftarrow{\$} \mathsf{SimEnc}(pk', \tau_1)$ for all i, $1 \le i \le r$; it then gives to $\mathcal{A}_2$ the tuple $(1^k, \{c_i^*\}, \mathsf{state}_2)$ and gets back $(c'_1, \ldots, c_q)$. For each c'_i, if there exists a j such that $c'_i = c_j^*$, S_2 sets $c_i := \mathsf{copy}_j$. Otherwise, it runs $\mathsf{SimExt}(\tau_2, c'_i)$ to get back (w, c_j^*, T). If $w \neq \perp$ then let $c_i := c'_i$; otherwise parse $T = (f_1, \ldots, f_n)$ and let $c_i := (j, f_1, \ldots, f_n)$.

We can now define $\mathcal{B}$ as follows: to start, $\mathcal{B}$ is given pk and can run $\mathcal{A}_1(pk)$ to obtain $(\mathcal{M}, \mathsf{state}_1, \mathsf{state}_2)$; if $\mathcal{A}_1$ makes any decryption queries, $\mathcal{B}$ can query its own D oracle and send the response back to $\mathcal{A}_1$. Next, $\mathcal{B}$ will sample $(m_1, \ldots, m_r) \xleftarrow{\$} \mathcal{M}$ and query its E oracle on these messages to obtain $(c_1^*, \ldots, c_r^*)$. It then runs $\mathcal{A}_2(1^k, c_1^*, \ldots, c_r^*, \mathsf{state}_2)$ to obtain $(c'_1, \ldots, c'_q)$. For each c'_i, if there exists a j such that $c'_i = c_j^*$ it sets $d_i := \mathsf{copy}_j$; otherwise it queries its D oracle on c'_i to obtain d_i. Finally, it sends $(\mathsf{state}_1, m_1, \ldots, m_r, d_1, \ldots, d_q)$ to $\mathcal{D}$. If $\mathcal{D}$ guesses this is the real

distribution then $\mathcal{B}$ guesses $b = 0$; otherwise, if $\mathcal{D}$ guesses this is the simulated distribution then $\mathcal{B}$ guesses $b = 1$.

If $b = 0$ in the CM-SSE game, then the input to $\mathcal{D}$ will be identical to that produced by the distribution Real; similarly, if $b = 1$ in the CM-SSE game, then the input to $\mathcal{D}$ will be identical to that produced by the distribution Sim. Thus, if D distinguishes between Real and Sim with non-negligible advantage, $\mathcal{B}$ succeeds in the CM-SSE game with the same non-negligible advantage. □

F Proof of CM-CCA Security (Theorem 5.2)

To prove the theorem, we need to demonstrate the indistinguishability of the following two games:

Game $\mathsf{CM\text{-}CCA}_0$
$(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $\sigma_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$
$pk^* \leftarrow (pk, \sigma_{\text{crs}})$
$b \xleftarrow{\$} \mathcal{A}^{E,D}(pk^*)$

Procedure $E(pk^*, m)$
$c \xleftarrow{\$} \mathsf{Enc}(pk, m)$
$\pi \xleftarrow{\$} \mathcal{P}(\sigma_{\text{crs}}, c, m)$
return (c, π)

Procedure $D(sk, c, \pi)$
return $\mathsf{Dec}(sk, c)$

Game $\mathsf{CM\text{-}CCA}_1$
$(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $(\sigma_{\text{crs}}, \tau_e) \xleftarrow{\$} E_1(1^k)$
$pk^* \leftarrow (pk, \sigma_{\text{crs}})$
$b \xleftarrow{\$} \mathcal{A}^{E,D}(pk^*)$

Procedure $E(pk^*, m)$
$r \xleftarrow{\$} \mathcal{M}$; $c \xleftarrow{\$} \mathsf{Enc}(pk, r)$
$\pi \xleftarrow{\$} S_2(\sigma_{\text{crs}}, \tau_s, c)$
add (m, c) to Q
return (c, π)

Procedure $D(sk, c, \pi)$
$(c', T) \leftarrow \mathsf{SimExt}(sk, \tau_e, c, \pi)$
if $\exists i$ s.t. $c' = c_i \in Q$ and $T \neq \perp$ return $T_m(m_i)$
else return $\mathsf{Dec}(sk, c)$

Procedure $\mathsf{SimExt}(sk, \tau_e, c, \pi)$
if $\mathcal{V}(\sigma_{\text{crs}}, (pk, c), \pi) = 0$ return $(\perp, \perp)$
$(m, c', T) \leftarrow \mathsf{Extract}(\tau_e, \pi)$
if $(c', T) \neq (\perp, \perp)$ but $T_c(c') \neq c$ or $T \notin \mathcal{T}$ return $(\perp, \perp)$
return (c', T)

We can do this by going through the following series of game transitions:

- Game G_0. Change to a simulated CRS and simulated proofs π in the E oracle; this is indistinguishable from $\mathsf{CM\text{-}CCA}_0$ by zero knowledge.
- Game G_1. Switch to using SimExt and Extract rather than Dec in the D oracle; this is indistinguishable from G_0 by the CM-SSE property of the NIZK.
- Game G_2. Switch the ciphertexts returned by the E oracle to be encryptions of random values; this is indistinguishable from G_1 by the IND-CPA security of the encryption scheme.
- Game G_3. Switch back to decrypting in the D oracle; this is now $\mathsf{CM\text{-}CCA}_1$ (and is indistinguishable from G_2 again by the CM-SSE property of the NIZK).

To start then, we switch the proofs in the encryption oracle from real to simulated:

Game $\mathsf{CM\text{-}CCA}_0$, $\boxed{G_0}$

1 $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $\sigma_{\mathrm{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$, $\boxed{(\sigma_{\mathrm{crs}}, \tau_s) \xleftarrow{\$} S_1(1^k)}$
2 $pk^* \leftarrow (pk, \sigma_{\mathrm{crs}})$
3 $b \xleftarrow{\$} \mathcal{A}^{E,D}(pk^*)$

Procedure $E(pk^*, m)$
4 $c \xleftarrow{\$} \mathsf{Enc}(pk, m)$
5 $\pi \xleftarrow{\$} \mathcal{P}(\sigma_{\mathrm{crs}}, c, m)$, $\boxed{\pi \xleftarrow{\$} S_2(\sigma_{\mathrm{crs}}, \tau_s, c)}$
6 return (c, π)

Procedure $D(sk, c, \pi)$
7 return $\mathsf{Dec}(sk, c)$

Lemma F.1. *If the NIZK satisfies zero knowledge, then Game G_0 is indistinguishable from* $\mathsf{CM\text{-}CCA}_0$.

Proof. To show this, we take an adversary $\mathcal{A}$ that distinguishes between $\mathsf{CM\text{-}CCA}_0$ and G_0 with some non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that can distinguish between an interaction with a simulator (S_1, S_2) and an interaction with the honest $(\mathsf{CRSSetup}, \mathcal{P})$ with the same non-negligible advantage. To start, $\mathcal{B}$ will receive some σ_{crs}. It then creates (pk, sk) itself and passes along $pk^* := (pk, \sigma_{\mathrm{crs}})$ to $\mathcal{A}$. On E oracle queries, $\mathcal{B}$ can simply form the ciphertext c honestly and query its own oracle to obtain a proof π; it will then return (c, π) to $\mathcal{A}$, while on D queries $\mathcal{B}$ can use sk to carry out decryption itself. At the end, if $\mathcal{A}$ guesses it is in $\mathsf{CM\text{-}CCA}_0$ $\mathcal{B}$ will guess it is interacting with the prover, while if $\mathcal{A}$ guesses it is in G_0 $\mathcal{B}$ will guess it is interacting with the simulator.

If $\mathcal{B}$ gets in an honest CRS and gets honest proofs from its oracle, then it is clear that it is executing the exact code of $\mathsf{CM\text{-}CCA}_0$; similarly, if it gets instead a simulated CRS and simulated proofs, it is executing the exact code of G_0. Interactions with $\mathcal{B}$ will therefore be identical to the interactions that $\mathcal{A}$ expects, so in particular its advantage will not change. Finally, as $\mathcal{B}$ successfully guesses its bit every time $\mathcal{A}$ does, we know that it will succeed with the same non-negligible advantage. □

Next, we can switch away from using sk in the decryption oracle; we do this by introducing SimExt and using $\mathsf{Extract}$ rather than Dec as follows:

Game G_0, $\boxed{G_1}$

1 $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $(\sigma_{\mathrm{crs}}, \tau_s) \xleftarrow{\$} S_1(1^k)$, $\boxed{(\sigma_{\mathrm{crs}}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)}$
2 $pk^* \leftarrow (pk, \sigma_{\mathrm{crs}})$
3 $b \xleftarrow{\$} \mathcal{A}^{E,D}(pk^*)$

Procedure $E(pk^*, m)$
4 $c \xleftarrow{\$} \mathsf{Enc}(pk, m)$
5 $\pi \xleftarrow{\$} S_2(\sigma_{\mathrm{crs}}, \tau_s, c)$
6 $\boxed{\text{add } (m, c) \text{ to } Q}$
7 return (c, π)

Procedure $D(sk, c, \pi)$
8 return $\mathsf{Dec}(sk, c)$, $\boxed{(c', T) \leftarrow \mathsf{SimExt}(sk, \tau_e, c, \pi)}$
9 $\boxed{\text{if } \exists i \text{ s.t. } c' = c_i \in Q \text{ and } T \neq \perp \text{ return } T_m(m_i)}$
10 $\boxed{\text{else return } \mathsf{Extract}(\tau_e, \pi)}$

Procedure $\mathsf{SimExt}(sk, \tau_e, c, \pi)$
11 if $\mathcal{V}(\sigma_{\mathrm{crs}}, (pk, c), \pi) = 0$ return $(\perp, \perp)$
12 $(m, c', T) \leftarrow \mathsf{Extract}(\tau_e, \pi)$
13 if $(c', T) \neq (\perp, \perp)$ but $T_c(c') \neq c$ or $T \notin \mathcal{T}$ return $(\perp, \perp)$
14 return (c', T)

Lemma F.2. *If the NIZK satisfies controlled-malleable simulation-sound extractability, then Game G_1 is indistinguishable from G_0.*

Proof. To see that the switch from decryption to extraction will go unnoticed, consider the event Fake in which $\mathcal{A}$ submits a D query (c, π) such that $\mathsf{Extract}(\tau_e, \pi) = (m, c', T)$ but either $m \neq \perp$ but $m \neq \mathsf{Dec}(sk, c)$ or $(c', T) \neq (\perp, \perp)$ but $c' \notin Q_c$, $c \neq T_c(c')$, or $T \notin \mathcal{T}$; it is clear that $\mathcal{A}$ can distinguish between G_0 and G_1 exactly when Fake occurs. To therefore show that this event can happen with at most negligible probability, we take an adversary $\mathcal{A}$ that can cause it to happen with some non-negligible probability ϵ and use it to construct an adversary $\mathcal{B}$ that breaks the CM-SSE property of the NIZK with related non-negligible probability ϵ'. The code for $\mathcal{B}$ is fairly straightforward: it will first be given $(\sigma_{\mathrm{crs}}, \tau_e)$ and proceed to generate (pk, sk) itself and give to $\mathcal{A}$ $pk^* := (pk, \sigma_{\mathrm{crs}})$. On E queries, $\mathcal{B}$ can simply encrypt the message and query S_2 to obtain a proof. On D queries for a ciphertext (c, π), $\mathcal{B}$ can first run $\mathsf{Extract}(\tau_e, \pi) = (m, c', T)$. If $m \neq \perp$, $\mathcal{B}$ can further compute $m' := \mathsf{Dec}(sk, c)$; if $m \neq m'$ then $\mathcal{B}$ can output the queried (c, π), and if $m = m'$ then $\mathcal{B}$ just continues with the game. In the case that $(c', T) \neq (\perp, \perp)$ instead, $\mathcal{B}$ can check to see if $c' \notin Q_c$, $c \neq T_x(c')$, or $T \notin \mathcal{T}$. If any of these hold $\mathcal{B}$ can once again output the queried (c, π); otherwise, $\mathcal{B}$ will again continue with the game. In the case that neither of these holds (i.e., $(m, c', T) = (\perp, \perp, \perp)$, $\mathcal{B}$ will again output (c, π).

Looking back at the CM-SSE definition, we can see that there are three possible winning conditions for this setting: (1) $m \neq \perp$ but $m \neq \mathsf{Dec}(sk, c)$, (2) $(c', T) \neq (\perp, \perp)$ but $c' \notin Q_c$, $c \neq T_x(c')$, or $T \notin \mathcal{T}$, or (3) $(m, c', T) = (\perp, \perp, \perp)$. Looking back at the code for $\mathcal{B}$, we can see that these are the exact conditions it is checking for; furthermore, looking back at the definition of Fake, we can see that these same conditions are being used here as well. In other words, each

case that would allow $\mathcal{A}$ to distinguish would also allow $\mathcal{B}$ to output a statement/proof pair (c, π) that met one of the CM-SSE winning conditions; this means that $\mathcal{B}$ will succeed in breaking the CM-SSE property with the same probability that $\mathcal{A}$ will cause Fake to occur (and thus distinguish between G_0 and G_1). □

Now that the proofs do not contain any information about the message and the secret key sk is not used anywhere, we can proceed to switch the message inside encryption queries to be random; this means changing Line 4 of G_1 as follows:

$$\begin{array}{rl} & \text{Game } G_1, \boxed{G_2} \\ 4 & c \xleftarrow{\$} \mathsf{Enc}(pk, m), \boxed{r \xleftarrow{\$} \mathcal{M};\ c \xleftarrow{\$} \mathsf{Enc}(pk, r)} \end{array}$$

Lemma F.3. *If the encryption scheme is IND-CPA secure, then Game G_2 is indistinguishable from G_1.*

Proof. To show this, we take an adversary $\mathcal{A}$ that distinguishes between the two games with some non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that can break the IND-CPA security of the encryption scheme with the same advantage ϵ.[20] To start, $\mathcal{B}$ will get in a public key pk; it can then generate $(\sigma_{\text{crs}}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$ and give to $\mathcal{A}$ $pk^* := (pk, \sigma_{\text{crs}})$. On E queries, $\mathcal{B}$ can pick a random message $r \xleftarrow{\$} \mathcal{M}$. It can then query its encryption oracle on (m, r) to get back a ciphertext c; it then fakes the proof π using τ_s and returns (c, π) to $\mathcal{A}$. On D queries, $\mathcal{B}$ can use τ_e to run SimExt and $\mathsf{Extract}$ and thus execute the code honestly. At the end of the game, if $\mathcal{A}$ guesses it is in G_1 then $\mathcal{B}$ will guess $b' = 0$ (indicating it is in the left world) and if $\mathcal{A}$ guesses G_2 then $\mathcal{B}$ will guess $b' = 1$.

To see that interactions with $\mathcal{B}$ will be indistinguishable from the honest interactions that $\mathcal{A}$ expects, we can consider the two cases for $\mathcal{B}$. If it receives an encryption of the left value, then it receives (and returns to $\mathcal{A}$) $\mathsf{Enc}(pk, m)$ and is executing the code for G_1. In the case that it receives an encryption of the right value, it is giving to $\mathcal{A}$ an encryption of a completely random value and is thus executing the code for G_2. $\mathcal{A}$ must therefore have the same advantage ϵ with $\mathcal{B}$ as it does in the real world; as $\mathcal{B}$'s guesses are correlated exactly with $\mathcal{A}$'s, we therefore see that $\mathcal{B}$ will succeed with the same advantage as $\mathcal{A}$. □

We can now switch back to using Dec again in the decryption oracle; this involves changing Line 10 to match $\mathsf{CM\text{-}CCA}_1$ above:

$$\begin{array}{rl} & \text{Game } G_2, \boxed{\mathsf{CM\text{-}CCA}_1} \\ 10 & \text{else return } \mathsf{Extract}(\tau_e, \pi), \boxed{\mathsf{Dec}(sk, c)} \end{array}$$

The proof of indistinguishability of G_2 and $\mathsf{CM\text{-}CCA}_1$ is analogous to the proof of Lemma F.2, as an adversary can distinguish between the two games only when the same event Fake occurs. As each game was indistinguishable from the previous game in the series, $\mathsf{CM\text{-}CCA}_0$ must be indistinguishable from $\mathsf{CM\text{-}CCA}_1$ and so we are done.

[20] Here we are considering a $\mathcal{B}$ that can make multiple queries to its IND-CPA oracle; the advantage for a $\mathcal{B}$ that was allowed to make only one would just be ϵ/q, where q is the number of queries $\mathcal{A}$ makes to E in the course of the game.

G Proof of Shuffle Security (Theorem 6.2)

To prove that the shuffle is compactly verifiable (as defined in Definition 6.1), we can go through the following series of games:

- Game G_0. The honest game for $b = 0$.
- Game G_1. In Step 1 we switch to using a simulated CRS σ'_{crs}, and in Step 2 we switch to using simulated proofs π in the initiation oracle. This is indistinguishable from G_0 by zero knowledge.
- Game G_2. In Step 1 we switch to using a simulated proof π in the regular shuffle oracle as well. This is indistinguishable from G_1 by strong derivation privacy (as defined in Definition 2.5).
- Game G_3. In Step 2, we switch to having both the initiation and shuffle oracles return fresh encryptions rather than re-randomizations; this is indistinguishable from G_2 by the re-randomizability of the encryption scheme.
- Game G_4. In Step 4, we extract the permutation φ from the proof π and return $\{\mathsf{Dec}(sk, \varphi(c_i))\}$. This is indistinguishable from G_2 by the CM-SSE property of the NIZK. (While CM-SSE does not guarantee that a permutation will be extracted directly, we argue in the proof of Lemma G.4 that even in the case that a instance x' and transformation T are recovered instead of a direct witness φ, a permutation can still be recovered.)
- Game G_4. In Step 1 we switch to using a trapdoor CRS σ_{crs}, and in Step 4 we use the extraction trapdoor to extract a witness m_i from each proof π_i for all i, $1 \leq i \leq n$, and use $\varphi(m_i)$ in place of $\mathsf{Dec}(sk, \varphi(c_i))$. This is indistinguishable from G_3 by the extractability property of the π_i.
- Game G_6. In Step 2, we switch to having both the initiation and shuffle oracles return encryptions of garbage; that is, for the values c'_i, rather than compute an honest shuffle they will instead pick random values $r_1, \ldots, r_n \xleftarrow{\$} \mathcal{M}$ and use $c'_i \xleftarrow{\$} \mathsf{Enc}(pk, r_i)$. This is indistinguishable from G_5 by the IND-CPA security of the encryption scheme.
- Game G_7. In Step 4, we pick a random permutation φ' and return $\{\varphi'(m_i)\}$. This is indistinguishable from G_6 by the CM-SSE property of the NIZK and the hardness of the relation R_{pk}.
- Game G_8. In Step 2, both oracles return to permuting and freshly encrypting for the ciphertexts rather than encrypting random values. This is indistinguishable from G_7 again by the IND-CPA security of the encryption scheme.
- Game G_9. In Step 4, we return to decrypting the ciphertexts c_i instead of using the values extracted from the proofs of knowledge; that is, we return $\{\mathsf{Dec}(sk, \varphi'(c_i))\}$. This is indistinguishable from G_8 again by the extractability property of the NIZKPoKs.
- Game G_{10}. In Step 2, both oracles return to performing an honest shuffle; i.e., re-randomizing the ciphertexts rather than performing fresh encryption. This is indistinguishable from G_9 again by the re-randomizability of the encryption scheme.
- Game G_{11}. In Step 2 we switch back to using honest proofs in the regular shuffle oracle. This is indistinguishable from G_{10} again by the derivation privacy of the NIZK.

- Game G_{12}. In Step 1 we switch back to an honest CRS, and in Step 2 we switch back to honest proofs in the initiation oracle. This is now the honest game for $b = 1$, and is indistinguishable from G_{11} again by zero knowledge.

Following this outline, the first step in our series of game transitions is to switch the proofs returned by the initiation oracle from honest proofs in G_0 to simulated proofs in G_1. In here and what follows, we use $\mathcal{R}$ to denote the randomness used for the encryption scheme, and $\mathcal{G}$ to denote the generator for the hard relation.

Game G_0, $\boxed{G_1}$

1 $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $\{(pk_i, sk_i)\} \xleftarrow{\$} \mathcal{G}(1^k)$; $S := \{pk_i\}$
2 $\sigma_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$
3 $\sigma'_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}'(1^k)$, $\boxed{(\sigma'_{\text{crs}}, \tau_s) \xleftarrow{\$} S_1(1^k)}$
4 $(\{c_i, \pi_i\}, \{c'_i\}, \pi, S' := \{pk_j\}) \xleftarrow{\$} \mathcal{A}^{\mathsf{InitShuffle},\mathsf{Shuffle}}(\sigma_{\text{crs}}, \sigma'_{\text{crs}}, pk, S)$
5 if $S \cap S' = \emptyset$ then abort
6 if $\mathcal{V}(\sigma_{\text{crs}}, c_i, \pi_i) = 0$ for any i then abort
7 if $\mathcal{V}'(\sigma'_{\text{crs}}, (\{c_i\}, \{c'_i\}, S'), \pi) = 0$ then abort
8 $b' \xleftarrow{\$} \mathcal{A}(\{\mathsf{Dec}(sk, c'_i)\})$

Procedure $\mathsf{InitShuffle}(\{c_i, \pi_i\}, pk_1)$
9 if $\mathcal{V}(\sigma_{\text{crs}}, c_i, \pi_i) = 0$ for any i then abort
10 $\varphi \xleftarrow{\$} S_n$; $R_1, \ldots, R_n \xleftarrow{\$} \mathcal{R}$
11 $c'_i \leftarrow \mathsf{ReRand}(pk, \varphi(c_i); R_i)$ for all i
12 $\pi \xleftarrow{\$} \mathcal{P}(\sigma'_{\text{crs}}, (\{c_i\}, \{c'_i\}, pk_1), (\varphi, \{R_i\}, sk_1))$, $\boxed{\pi \xleftarrow{\$} S_2(\sigma'_{\text{crs}}, \tau_s, (\{c_i\}, \{c'_i\}, pk_1))}$
13 output $(\{c'_i\}, \pi, \{pk_1\})$

Procedure $\mathsf{Shuffle}(\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\}, pk_k)$
14 if $\mathcal{V}(\sigma_{\text{crs}}, c_i, \pi_i) = 0$ for any i then abort
15 if $\mathcal{V}'(\sigma'_{\text{crs}}, (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi) = 0$ then abort
16 $\varphi \xleftarrow{\$} S_n$; $R_1, \ldots, R_n \xleftarrow{\$} \mathcal{R}$
17 $c''_i \leftarrow \mathsf{ReRand}(pk, \varphi(c'_i); R_i)$ for all i
18 $\pi' \xleftarrow{\$} \mathsf{ZKEval}(\sigma'_{\text{crs}}, T := (\varphi, \{R_i\}, (sk_k, pk_k), \emptyset), (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi))$
19 output $(\{c''_i\}, \pi', \{pk_j\} \cup pk_k)$

Lemma G.1. *If the proof π is zero knowledge, Game G_1 is indistinguishable from G_0.*

Proof. To show this, we can take an adversary $\mathcal{A}$ that distinguishes between G_0 and G_1 with some non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that can distinguish between an honest CRS followed by interactions with an honest prover and a simulated CRS and interactions with a simulator with the same advantage ϵ. To start, $\mathcal{B}$ will have access to some CRS σ'_{crs}; it will then create (pk, sk), $\{(pk_i, sk_i)\}$, and σ_{crs} by itself and give all the public values to $\mathcal{A}$. On queries to the initiation oracle, $\mathcal{B}$ will come up with an honestly chosen permutation $\varphi \xleftarrow{\$} S_n$ and randomness $\{R_i\}$ and compute the shuffle $\{c'_i\}$ honestly. It can then query its own oracle to get back a proof π; it will then pass back this information to $\mathcal{A}$. On queries to the regular shuffle oracle, $\mathcal{B}$ will simply

execute the code honestly. At the end of the game, $\mathcal{B}$ will guess that it is interacting with a prover if $\mathcal{A}$ guesses G_0 and that it is interacting with a simulator if $\mathcal{A}$ guesses G_1.

To see that interactions with $\mathcal{B}$ will appear the same as interactions with the honest games, we just observe that when $\mathcal{B}$ is handed an honest σ'_{crs} and access to an honest prover it is executing the exact code of Game G_0; similarly, when it is given a simulated σ'_{crs} and access to a simulator, it is executing the exact code of Game G_1. The advantage of $\mathcal{A}$ in guessing the game will then be the same as its advantage when guessing with $\mathcal{B}$; as $\mathcal{B}$ is right every time $\mathcal{A}$ is right, we can conclude that $\mathcal{B}$ will guess correctly with non-negligible advantage ϵ as well. □

Next, we switch in Game G_2 to using simulated proofs in the regular shuffle oracle as well; this means changing only one line of G_1 as follows:

Game G_1, $\boxed{G_2}$

18 $\pi' \xleftarrow{\$} \mathsf{ZKEval}(\sigma'_{\text{crs}}, T := (\varphi, \{R_i\}, (sk_k, pk_k), \emptyset), ((\{c_i\}, \{c'_i\}, \{pk_j\}), \pi))$,

$\boxed{\pi' \xleftarrow{\$} S_2(\sigma'_{\text{crs}}, \tau_s, (\{c_i\}, \{c''_i\}, \{pk_j\} \cup pk_k))}$

Lemma G.2. *If the proof π is strongly derivation private, Game G_2 is indistinguishable from G_1.*

Proof. To show this, we will proceed through a series of hybrids H_0 through H_q, where q is some upper bound on the number of queries $\mathcal{A}$ can make to the regular shuffle oracle. In the hybrid H_i, the first i regular shuffle queries will be answered using ZKEval, while the last ones (at most $q - i$) will be answered using the simulator; it is clear then that H_0 is identical to G_1 and H_q is identical to G_2, so that if we show the indistinguishability of H_i from H_{i+1} for all i, $1 \leq i < q$, then we prove the lemma.

To do this, we take an adversary that can distinguish between H_i and H_{i+1} with some non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that can distinguish between proofs from ZKEval and simulated proofs with the same advantage ϵ. To start, $\mathcal{B}$ will have access to some CRS σ'_{crs} and simulation trapdoor τ_s; it will then create (pk, sk), $\{(pk_i, sk_i)\}$, and σ_{crs} by itself and give all the public values to $\mathcal{A}$. On queries to the initiation oracle, $\mathcal{B}$ will come up with an honestly chosen permutation $\varphi \xleftarrow{\$} S_n$ and randomness $\{R_i\}$ and compute the shuffle $\{c'_i\}$ honestly. It can then use τ_s to simulate a proof π and pass all this information back to $\mathcal{A}$. On queries to the regular shuffle oracle using public key pk_k, $\mathcal{B}$ will again pick an honest permutation $\varphi \xleftarrow{\$} S_n$ and randomness $\{R_i\}$ to compute the shuffle; it will also set $T := (\varphi, \{R_i\}, \{sk_k, pk_k\}, \emptyset)$. For the first i queries of the form $(\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\}, pk_k)$, $\mathcal{B}$ will return $\pi' \xleftarrow{\$} \mathsf{ZKEval}(\sigma'_{\text{crs}}, T, (x := (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi))$. For the $i+1$-st query, $\mathcal{B}$ will query its own oracle on input (x, π) to get back a proof π' that it then returns to $\mathcal{A}$. Finally, for the rest of the queries, $\mathcal{B}$ will return $\pi' \xleftarrow{\$} S_2(\sigma'_{\text{crs}}, \tau_s, x)$. At the end of the game, $\mathcal{B}$ will guess that its proof came from ZKEval if $\mathcal{A}$ guesses H_i and the simulator if $\mathcal{A}$ guesses H_{i+1}.

To see that interactions with $\mathcal{B}$ will appear the same as interactions with the honest games, we just observe that when $\mathcal{B}$ is given a proof from ZKEval it is executing the exact code of Game H_i; similarly, when it is given a simulated proof, it is executing the exact code of Game H_{i+1}. The advantage of $\mathcal{A}$ in guessing the game will then be the same as its advantage when guessing with $\mathcal{B}$; as $\mathcal{B}$ is right every time $\mathcal{A}$ is right, we can conclude that $\mathcal{B}$ will guess correctly with non-negligible advantage ϵ as well. □

Next, in Game G_3 we switch to computing fresh encryptions of the given ciphertexts rather than re-randomizing them. This means changing two lines from G_2 as follows:

Game G_2, $\boxed{G_3}$

14 $c'_i \leftarrow \mathsf{ReRand}(pk, \varphi(c_i); R_i)$ for all i, $\boxed{c'_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi(c_i)))\text{ for all } i}$

21 $c''_i \leftarrow \mathsf{ReRand}(pk, \varphi(c'_i); R_i)$ for all i, $\boxed{c''_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi(c'_i)))\text{ for all } i}$

Lemma G.3. *If the encryption scheme is re-randomizable, then Game G_3 is indistinguishable from G_2.*

Proof. To show this, we will proceed through a series of hybrids H_0 through H_q, where q is some upper bound on the number of queries $\mathcal{A}$ can make to the shuffle oracles. In the hybrid H_i, the first i shuffle queries will be answered using ReRand, while the last ones (at most $q-i$) will be answered using fresh encryptions; it is clear then that H_0 is identical to G_2 and H_q is identical to G_3, so that if we show the indistinguishability of H_i from H_{i+1} for all i, $1 \leq i < q$, then we prove the lemma.

To do this, we take an adversary that can distinguish between H_i and H_{i+1} with some non-negligible advantage ϵ and use it to construct an adversary $\mathcal{B}$ that can distinguish between fresh and re-randomized encryptions with the same advantage ϵ. To start, $\mathcal{B}$ will have access to some keypair (pk, sk); it will then create $\{(pk_i, sk_i)\} \xleftarrow{\$} \mathcal{G}(1^k)$, $\sigma_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$, and $(\sigma'_{\text{crs}}, \tau_s) \xleftarrow{\$} S_1(k)$ by itself and give all the public values to $\mathcal{A}$. On the first i queries, $\mathcal{B}$ can act as either the initiation or the shuffle oracle: for the initiation oracle, $\mathcal{B}$ will come up with an honestly chosen permutation $\varphi \xleftarrow{\$} S_n$ and randomness and compute the shuffle $\{c'_i\}$ honestly. It can then use τ_s to simulate a proof π and pass all this information back to $\mathcal{A}$. On queries to the regular shuffle oracle using public key pk_k, $\mathcal{B}$ will again pick an honest permutation $\varphi \xleftarrow{\$} S_n$ and randomness $\{R_i\}$ to compute the shuffle; it will then again use τ_s to simulate the proof π. For the $i+1$-st query, if the query is to the initiation oracle then $\mathcal{B}$ will query its own oracle on input $\{c_i\}$, and if the query is to the regular shuffle oracle then $\mathcal{B}$ will query on input $\{c'_i\}$.[21] It will then get in response a set of ciphertexts $\{c''_i\}$ and return these to $\mathcal{A}$, along with the appropriate proof and public key values (so if the query was to the initiation oracle it sends $(\{c''_i\}, \pi, \{pk_1\})$, where π is a simulated proof, and if the query was to the shuffle oracle with public key pk_k it sends $(\{c''_i\}, \pi', \{pk_j\} \cup pk_k)$, where π' is again simulated). Finally, for the rest of the queries $\mathcal{B}$ will compute all the c'_i (respectively c''_i) as $\mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi(c_i)))$ and return these to $\mathcal{A}$. At the end of the game, $\mathcal{B}$ will guess that it is getting freshly encrypted ciphertexts if $\mathcal{A}$ guesses H_i and re-randomized ciphertexts if $\mathcal{A}$ guesses H_{i+1}.

To see that interactions with $\mathcal{B}$ will appear the same as interactions with the honest games, we just observe that when $\mathcal{B}$ is given a set of freshly encrypted ciphertexts it is executing the exact code of Game H_i; similarly, when it is given a set of re-randomized ciphertexts, it is executing the exact code of Game H_{i+1}. The advantage of $\mathcal{A}$ in guessing the game will then be the same as its advantage when guessing with $\mathcal{B}$; as $\mathcal{B}$ is right every time $\mathcal{A}$ is right, we can conclude that $\mathcal{B}$ will guess correctly with non-negligible advantage ϵ as well. □

Next, we switch in Game G_4 to returning $\{\mathsf{Dec}(sk, \varphi(c_i))\}$ rather than $\{\mathsf{Dec}(sk, c'_i)\}$, where φ is the permutation used to obtain $\{c'_i\}$ from $\{c_i\}$ (and which we get from the proof π).

[21] Note that in the game in Definition 2.8 we currently consider only the case of a single ciphertext; by a simple hybrid argument, however, we can extend this to the case of n ciphertexts, where either all are re-randomized or all are fresh encryptions.

Game G_3, $\boxed{G_4}$

$(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$; $\{pk_i, sk_i)\} \xleftarrow{\$} \mathcal{G}(1^k)$; $S := \{pk_i\}$

$\sigma_{\mathrm{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$

$(\sigma'_{\mathrm{crs}}, \tau_s) \xleftarrow{\$} S_1(1^k)$, $\boxed{(\sigma'_{\mathrm{crs}}, \tau_s, \tau'_e) \xleftarrow{\$} SE_1(1^k)}$

$(\{c_i, \pi_i\}, \{c'_i\}, \pi, S' := \{pk_j\}) \xleftarrow{\$} \mathcal{A}^{\mathsf{InitShuffle},\mathsf{Shuffle}}(\sigma_{\mathrm{crs}}, \sigma'_{\mathrm{crs}}, pk, S)$

if $S \cap S' = \emptyset$ then abort

if $\mathcal{V}(\sigma_{\mathrm{crs}}, c_i, \pi_i) = 0$ for any i then abort

if $\mathcal{V}'(\sigma'_{\mathrm{crs}}, (\{c_i\}, \{c'_i\}, S'), \pi) = 0$ then abort

$b' \xleftarrow{\$} \mathcal{A}(\{\mathsf{Dec}(sk, c'_i)\})$, $\boxed{(w, x', T) \leftarrow \mathsf{Extract}(\tau'_e, \pi)}$

$\boxed{\text{if } w \neq \perp \text{ then parse } w = (\varphi', \{R_i\}, \{sk_j\});\ b' \xleftarrow{\$} \mathcal{A}(\{\mathsf{Dec}(sk, \varphi'(c_i))\})}$

$\boxed{\text{else parse } (x', T) = ((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), (\varphi', \{R_i\}, \{sk_k, pk_k\}, \{pk'_k\}))}$

$\boxed{\text{if } ((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi}) \in Q \text{ then } b' \xleftarrow{\$} \mathcal{A}(\{\mathsf{Dec}(sk, \varphi' \circ \hat{\varphi}(c_i))\})}$

Procedure $\mathsf{InitShuffle}(\{c_i, \pi_i\}, pk_1)$

if $\mathcal{V}(\sigma_{\mathrm{crs}}, c_i, \pi_i) = 0$ for any i then abort

$\varphi \xleftarrow{\$} S_n$; $R_1, \ldots, R_n \xleftarrow{\$} \mathcal{R}$

$c'_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi(c_i)))$ for all i

$\pi \xleftarrow{\$} S_2(\sigma'_{\mathrm{crs}}, \tau_s, (\{c_i\}, \{c'_i\}, pk_1))$

$\boxed{\text{add } ((\{c_i\}, \{c'_i\}, \{pk_1\}), \varphi) \text{ to } Q}$

output $(\{c'_i\}, \pi, \{pk_1\})$

Procedure $\mathsf{Shuffle}(\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\}, pk_k)$

if $\mathcal{V}(\sigma_{\mathrm{crs}}, c_i, \pi_i) = 0$ for any i then abort

if $\mathcal{V}'(\sigma'_{\mathrm{crs}}, (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi) = 0$ then abort

$\varphi \xleftarrow{\$} S_n$; $R_1, \ldots, R_n \xleftarrow{\$} \mathcal{R}$

$c''_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi(c'_i)))$ for all i, $\boxed{(w, x', T) \leftarrow \mathsf{Extract}(\tau_e, \pi)}$

$\boxed{\text{if } w \neq \perp \text{ then parse } w = (\varphi', \{R_i\}, \{sk_j\});\ c''_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi \circ \varphi'(c_i))) \text{ for all } i}$

$\boxed{\text{else parse } (x', T) = ((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), (\varphi', \{R_i\}, \{sk_k, pk_k\}, \{pk'_k\}))}$

$\boxed{\text{if } ((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi}) \in Q \text{ then } c''_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi \circ \varphi' \circ \hat{\varphi}(c_i))) \text{ for all } i}$

$\pi' \xleftarrow{\$} S_2(\sigma'_{\mathrm{crs}}, \tau_s, (\{c_i\}, \{c''_i\}, \{pk_j\} \cup pk_k))$

$\boxed{\text{add } ((\{c_i\}, \{c''_i\}, \{pk_j\} \cup pk_k), \varphi \circ \varphi' \circ \hat{\varphi}) \text{ to } Q}$

output $(\{c''_i\}, \pi', \{pk_j\} \cup pk_k)$

Lemma G.4. *If the NIZK π satisfies controlled-malleable simulation-sound extractability (as defined in Definition 3.1), Game G_4 is indistinguishable from G_3.*

Proof. To show this, we will argue that the probability that an adversary $\mathcal{A}$ is able to cause games G_3 and G_4 to behave differently is at most negligible; to do this, we can take an adversary $\mathcal{A}$ that

causes this event with some non-negligible probability ϵ and use it to produce an adversary $\mathcal{B}$ that can win at the CM-SSE game with related non-negligible probability ϵ'.

To start, $\mathcal{B}$ gets in σ_{crs} and τ_e; it can then honestly generate all the other parameters itself and pass these along to $\mathcal{A}$. We will argue that, throughout the game, the database Q stores only tuples of the form $((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi})$ where $\mathsf{Dec}(sk, \hat{c}_i) = \mathsf{Dec}(sk, \hat{\varphi}(c_i))$ for all i.

On $\mathsf{InitShuffle}$ queries, $\mathcal{B}$ will perform the shuffle honestly according to game G_4, making use of the database Q to store the permutations, and then query its S_2 oracle to obtain the corresponding proof. Note that the output of the query will be identical to that in game G_3, and that by correctness of the encryption scheme, our invariant on Q must hold for all entries added in these queries.

Eventually, $\mathcal{A}$ will either query the regular shuffle oracle on a tuple $(\{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\})$ or output such a tuple at the end (i.e., in line 4). Given such a tuple, $\mathcal{B}$ can now compute $(w, x', T) := \mathsf{Extract}(\tau_e, \pi)$. If $(w, x', T) = (\perp, \perp, \perp)$, then $\mathcal{B}$ can immediately output $(x := (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi)$. Otherwise, if $w \neq \perp$ but w has the wrong structure (i.e., $w \neq (\varphi', \{R_i\}, \{sk_j\})$) then $\mathcal{B}$ can once again output (x, π). If, on the other hand, $w = (\varphi', \{R_i\}, \{sk_j\})$, $\mathcal{B}$ can check that $\mathsf{Dec}(sk, c'_i) = \mathsf{Dec}(sk, \varphi'(c_i))$ for all i; if this equality does not hold for some i, $\mathcal{B}$ can once again output (x, π). If it does hold, then, if we are in step 4, $\mathcal{B}$ can send $\mathcal{A}$ $\{\mathsf{Dec}(sk, \varphi'(c_i))\} = \{\mathsf{Dec}(sk, c'_i)\}$, which will be identical to the output in both game G_3 and game G_4, and the tuple added to Q will satisfy the invariant. Similarly, if this is a shuffle query, then $\mathcal{B}$ can proceed as in Game G_4; again, this will be identical to the response it would give in G_3, and we are guaranteed that the value added to Q will satisfy the invariant.

If $(x', T) \neq (\perp, \perp)$, $\mathcal{B}$ can check to see that these values have the proper structure and x' was output by one of the shuffle oracle (i.e., queried to the simulator at some point), if not then $\mathcal{B}$ can once again output the pair (x, π). Otherwise, $\mathcal{B}$ can look up in Q the permutation $\hat{\varphi}$ used at the time x' was generated; note that by our invariant it will be be the case that $\mathsf{Dec}(sk, \hat{\varphi}(c_i)) = \mathsf{Dec}(sk, \hat{c}_i)$ for all i. Then $\mathcal{B}$ can finally check if $\mathsf{Dec}(sk, c'_i) = \mathsf{Dec}(sk, \varphi'(\hat{c}_i))$ for all i and output (x, π) if equality doesn't hold for some i. If none of these cases hold, then it will be the case that $\mathsf{Dec}(sk, c'_i) = \mathsf{Dec}(sk, \varphi'(\hat{c}_i)) = \mathsf{Dec}(sk, \varphi' \circ \hat{\varphi}(c_i))$. Thus, $\mathcal{B}$ can proceed as in game G_4 and this will be identical to the responses it would give in G_3.

If $\mathcal{B}$ reaches the end of the game without finding a (x, π) tuple to output, then it will output $\perp$ to indicate failure. Note that if this happens, then this means $\mathcal{B}$'s behavior throughout was identical to the honest behavior in both G_3 and G_4.

Similarly, each of the cases in which $\mathcal{B}$ does output (x, π) correspond to one of the winning conditions for the CM-SSE game; if $(w, x', T) = (\perp, \perp, \perp)$ (or are otherwise improperly formatted) then we are in case (3); if $w \neq \perp$ but $\mathsf{Dec}(sk, c'_i) \neq \mathsf{Dec}(sk, \varphi(c_i))$ for some i then we are in case (1); if $(x', T) \neq (\perp, \perp)$ but x' was never queried then we are in the first part of case (2); if $T \neq (\varphi', \{R_i\}, \{pk_j\})$ then we are in the third part of case (2); finally, if $\mathsf{Dec}(sk, c'_i) \neq \mathsf{Dec}(sk, \varphi' \circ \hat{\varphi}(c_i))$ for some i then we are in the second part of case (2). As the case in which $\mathcal{A}$ causes the games to differ therefore implies that $\mathcal{B}$ can win the CM-SSE game, we know that $\mathcal{B}$ will succeed with the same probability as $\mathcal{A}$ does. □

Next, in Game G_5 we switch to using the values extracted from the proofs π_i rather than the plaintexts recovered from decrypting the c_i; we do this by changing six lines from G_4 as follows:

Game G_4, $\boxed{G_5}$

2 $\sigma_{\text{crs}} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$, $\boxed{(\sigma_{\text{crs}}, \tau_e) \xleftarrow{\$} E_1(1^k)}$

9 if $w \neq \perp$ then parse $w = (\varphi', \{R_i\}, \{sk_j\})$; $b' \xleftarrow{\$} \mathcal{A}(\{\mathsf{Dec}(sk, \varphi'(c_i))\})$, $\boxed{b' \xleftarrow{\$} \mathcal{A}(\{E_2(\tau_e, \varphi'(\pi_i))\})}$

11 if $((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi}) \in Q$ then $b' \xleftarrow{\$} \mathcal{A}(\{\mathsf{Dec}(sk, \varphi' \circ \hat{\varphi}(c_i))\})$, $\boxed{b' \xleftarrow{\$} \mathcal{A}(\{E_2(\tau_e, \varphi' \circ \hat{\varphi}(\pi_i))\})}$

14 $c'_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi(c_i)))$ for all i, $\boxed{c'_i \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi(\pi_i)))$ for all $i}$

22 if $w \neq \perp$ then parse $w = (\varphi', \{R_i\}, \{sk_j\})$; $c''_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi \circ \varphi'(c_i)))$ for all i,
$\boxed{c''_i \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi \circ \varphi'(\pi_i)))$ for all $i}$

24 if $((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi}) \in Q$ then $c''_i \xleftarrow{\$} \mathsf{Enc}(pk, \mathsf{Dec}(sk, \varphi \circ \varphi' \circ \hat{\varphi}(c_i)))$ for all i,
$\boxed{c''_i \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi \circ \varphi' \circ \hat{\varphi}(\pi_i)))$ for all $i}$

Lemma G.5. *If the NIZKPoKs π_i satisfies soundness, Game G_5 is indistinguishable from G_4.*

Proof. For an adversary to notice a different between these two games, there must be at least one index i for which the decryption of c_i differed from the value extracted from π_i. As the only valid witness for π_i is the value in c_i, however, this would imply that $\mathcal{A}$ had produced a valid proof of a false statement, which violates the soundness of the proof system (and in particular the extractability property). This event must therefore happen with negligible probability for a given index i (and therefore by a union bound for any index), and thus the two games are indistinguishable. □

Next, in Game G_6 we switch the values in the ciphertexts returned by queries to the shuffle oracles; rather than performing an honest shuffle, both shuffle oracles will now return encryptions of completely random values instead.

Game G_5, $\boxed{G_6}$

14 $c'_i \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi(\pi_i)))$ for all i, $\boxed{(r_1, \ldots, r_n) \xleftarrow{\$} \mathcal{M}^n;\ c'_i \xleftarrow{\$} \mathsf{Enc}(pk, r_i)$ for all $i}$

22 if $w \neq \perp$ then parse $w = (\varphi', \{R_i\}, \{sk_j\})$; $c''_i \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi'(\pi_i)))$ for all i,
$\boxed{(r_1, \ldots, r_n) \xleftarrow{\$} \mathcal{M}^n;\ c''_i \xleftarrow{\$} \mathsf{Enc}(pk, r_i)$ for all $i}$

24 if $((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi}) \in Q$ then $c''_i \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi \circ \varphi' \circ \hat{\varphi}(\pi_i)))$ for all i,
$\boxed{(r_1, \ldots, r_n) \xleftarrow{\$} \mathcal{M}^n;\ c''_i \xleftarrow{\$} \mathsf{Enc}(pk, r_i)$ for all $i}$

Lemma G.6. *If the encryption scheme is IND-CPA secure, Game G_6 is indistinguishable from G_5.*

Proof. To see that these games are indistinguishable, we can consider a series of hybrids H_i. In each game H_i, the first i ciphertexts c'_j returned by the shuffle oracle are encryptions of random values, while the ones from $i+1$ on are computed as $c'_k \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi(\pi_k)))$ for all k, $i+1 \leq k \leq n$ (and for the appropriately extracted permutation); we can then see that H_0 is just the game G_5 and H_n is G_6.

To see that H_i must be indistinguishable from H_{i+1}, we can consider an adversary $\mathcal{A}$ that succeeds with non-negligible advantage ϵ in distinguishing between H_i and H_{i+1} and use it to

construct an adversary $\mathcal{B}$ that succeeds in attacking the IND-CPA security of the encryption scheme with related non-negligible advantage ϵ'; for simplicity, we assume that $\mathcal{A}$ makes only one oracle query, but our result does generalize to the case of multiple queries.[22]

To start, $\mathcal{B}$ will get a public key pk. It can then generate all the other parameters for the shuffle on its own and pass these along to $\mathcal{A}$; in particular, $\mathcal{B}$ will generate $(\sigma_{\text{crs}}, \tau_e) \xleftarrow{\$} E_1(1^k)$ and $(\sigma'_{\text{crs}}, \tau_s, \tau'_e) \xleftarrow{\$} SE_1(1^k)$. On a shuffle query (of either type), $\mathcal{B}$ will first simulate the proof π. It can then either pick a random permutation φ (for the initiation oracle) or extract/compose a permutation φ using τ'_e (for the shuffle oracle), in addition to random values $r_1, \ldots, r_i \xleftarrow{\$} \mathcal{M}$. It then sets $c'_j \xleftarrow{\$} \mathsf{Enc}(pk, r_j)$ for all j, $1 \leq j \leq i$, and $c'_k \xleftarrow{\$} \mathsf{Enc}(pk, E_2(\tau_e, \varphi(\pi_k)))$ for all k, $i+2 \leq k \leq n$. For the $i+1$-st encryption, $\mathcal{B}$ can use τ_e to extract the message m from $\varphi(\pi_{i+1})$ and pick a random value $r \xleftarrow{\$} \mathcal{M}$. $\mathcal{B}$ will now query its own IND-CPA oracle on the message pair (m, r) and get back some ciphertext c; it will then use $c'_{i+1} := c$. At the end of the game, it will output the same guess bit as $\mathcal{A}$.

To see that $\mathcal{B}$ will succeed with the same advantage as $\mathcal{A}$, we must first show that interactions with $\mathcal{B}$ are indistinguishable from interactions in the honest games. In the case that its encryption oracle returns an encryption of the message m, $\mathcal{B}$ will then be using $c'_{i+1} = \mathsf{Enc}(pk, E_2(\tau_e, \varphi(\pi_{i+1})))$ and thus executing the exact code for H_i. Similarly, if the encryption oracle returns instead an encryption of r, $\mathcal{B}$ will be using an encryption of a random value as c'_{i+1} and thus executing the exact code for H_{i+1}; in either case, therefore, $\mathcal{B}$ is executing the code for one of the two games and so interactions with $\mathcal{B}$ are indistinguishable from interactions with the honest game. In addition, $\mathcal{B}$ succeeds every time $\mathcal{A}$ does, so if $\mathcal{A}$ succeeds with non-negligible advantage then $\mathcal{B}$ does as well. □

Next, in Game G_7 we switch to using a random permutation at the end rather than the permutation extracted from $\mathcal{A}$'s proof. This means replacing lines 8-11 from G_6 as follows:

	Game G_6	Game G_7
8	$(w, x', T) \leftarrow \mathsf{Extract}(\tau'_e, \pi)$	$\varphi \xleftarrow{\$} S_n$
9	if $w \neq \perp$ then parse $w = (\varphi', \{R_i\}, \{pk_j\})$; $b' \xleftarrow{\$} \mathcal{A}(\{E_2(\tau_e, \varphi'(\pi_i))\})$	$b' \xleftarrow{\$} \mathcal{A}(\{E_2(\tau_e, \varphi(\pi_i))\})$
10	else parse $(x', T) = ((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), (\varphi', \{R_i\}, \{sk_k, pk_k\}, \{pk'_k\}))$	
11	if $((\{c_i\}, \{\hat{c}_i\}, \{pk_j\}), \hat{\varphi}) \in Q$ then $b' \xleftarrow{\$} \mathcal{A}(\{E_2(\tau_e, \varphi' \circ \hat{\varphi}(\pi_i))\})$	

Lemma G.7. *If the NIZK π is CM-SSE and R_{pk} is a hard relation, Game G_7 is indistinguishable from G_6.*

Proof. First we note that in Game G_6, if the adversary is called on Step 11, then it is given as input a set of messages shuffled according to $\varphi' \circ \hat{\varphi}$, where $\hat{\varphi}$ was retrieved from Q. As we saw back in Game G_4, permutations are added to Q when the adversary queries $\mathsf{InitShuffle}$ or $\mathsf{Shuffle}$. In both of these cases the permutation is chosen at random (and in the case of $\mathsf{Shuffle}$ queries, composed with some other permutations, resulting in another random permutation) and never affects any response given to the adversary (until Step 11). Thus, in this case we are using a permutation which is completely random in the adversary's view, so this must be identical to what the adversary is given in Step 9 of Game G_7.

We observe then, that an adversary $\mathcal{A}$ that can distinguish between the games must cause one of the following three events to occur (we refer to the event in which one of these occurs as Fake): (1) the extractor produces $w \neq \perp$, (2) the extractor produces an $x' = (\{c_i\}, \{\hat{c}_i\}, \{pk_j\})$ that was

[22] In the case of multiple queries, say q, $\mathcal{B}$'s advantage will be ϵ/q, which is still non-negligible for polynomial q.

never stored in Q, or (3) the extractor fails entirely and produces invalid (x', T). We will show that, if $\mathcal{A}$ can cause Fake to occur with some non-negligible probabiliy ϵ then we can use it to construct an adversary $\mathcal{B}$ that can either win at the CM-SSE game or break the hardness of the relation with related non-negligible probability.

With this in mind, we can consider the tuple $(\{c_i, \pi_i\}, \{c'_i\}, \pi, S' := \{pk_j\})$ output by $\mathcal{A}$, and consider our adversary $\mathcal{B}$, who is given $(\sigma_{\text{crs}}, \tau_e)$ and a value pk_j and is asked to either break the CM-SSE property of the NIZK (acting in a strategy we refer to as $\mathcal{B}_1$) or break the hard relation; i.e., produce a value sk_j such that $(pk_j, sk_j) \in R_{pk}$ (in a strategy we refer to as $\mathcal{B}_2$). To create an input for $\mathcal{A}$, $\mathcal{B}_1$ will ignore pk_j and generate on its own encryption keys (pk, sk) and a set $S := \{pk_i\}$ of mix server keys (along with the corresponding values sk_i such that $(pk_i, sk_i) \in R_{pk}$). $\mathcal{B}_1$ can next answer shuffle queries by first encrypting random values, and then querying its S_2 oracle to get back a proof.

The adversary $\mathcal{B}_2$, on the other hand, will ignore $(\sigma_{\text{crs}}, \tau_e)$. To create input for $\mathcal{A}$, $\mathcal{B}_2$ will instead generate $(\sigma_{\text{crs}}, \tau_s, \tau_e)$ and (pk, sk) on its own; it will then also generate a set $S_{\mathcal{B}_2} := \{pk_i\}$ of random mix server keys and give to $\mathcal{A}$ $S := S_{\mathcal{B}_2} \cup pk_j$ (as well as the other parameters it generated). To answer shuffle queries, $\mathcal{B}_2$ can just simulate the proofs using τ_s.

As the behavior of both $\mathcal{B}_1$ and $\mathcal{B}_2$ is indistinguishable from the honest game interactions, in particular $\mathcal{A}$ cannot tell when it is interacting with $\mathcal{B}_1$ or when it is interacting with $\mathcal{B}_2$. $\mathcal{B}$'s strategy can then be to randomly pick which path to pursue at the start; as the bit it picks will be independent of $\mathcal{A}$'s view, it will pick the correct path (i.e., whether $\mathcal{A}$ will break the CM-SSE property or the hard relation) with probability 1/2.

Finally $\mathcal{A}$ outputs its tuple in Step 4. If $S' \cap S = \emptyset$ or any of the proofs fail to verify then $\mathcal{A}$ is not behaving properly within the game and so both $\mathcal{B}_1$ and $\mathcal{B}_2$ can simply abort. Otherwise, we can look at $(w, x', T) := \mathsf{Extract}(\tau_e, \pi)$. If $w \neq \perp$ and $w = (\varphi', \{R_i\}, \{sk_j\})$, then either $(pk_i, sk_i) \in R_{pk}$ for all i or there is some i such that this does not hold. In the former case, and in particular if $(pk_j, sk_j) \in R_{pk}$ for the pk_j that $\mathcal{B}_2$ started with, then $\mathcal{B}_2$ can output sk_j to win its game. As the pk_j that $\mathcal{B}_2$ was given is distributed identically to the ones that it generated itself, this will happen with probability $|S' \cap S|/|S| \geq 1/|S|$ (i.e., as long as pk_j was included in S') and so $\mathcal{B}_2$ will succeed with overall probability $\epsilon/|S|$. In the latter case, $\mathcal{A}$ has managed to prove a false statement for the i such that $(pk_i, sk_i) \notin R_{pk}$ and so $\mathcal{B}_1$ can output the tuple $(x := (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi)$ to win the CM-SSE game.

If instead of the cases above it holds that $(x', T) \neq (\perp, \perp)$, we can check to see if x' was in fact queried to the simulated proof oracle as part of one of the shuffle queries. If this check fails then $\mathcal{B}_1$ can again output (x, π) to win its game; if it succeeds, then x' must have been stored in Q as part of the same shuffle query and so, as discussed above, the games will be identical.

The only remaining case is the one in which Step 11 is executed correctly, and we have argued that in this case the two games are identical. Thus $\mathcal{B}$ can succeed (using either $\mathcal{B}_1$ or $\mathcal{B}_2$) with probability at least $1/|S|$ whenever $\mathcal{A}$ succeeds and it guesses the correct path; it thus succeeds with overall probability at least $\epsilon/2|S|$. □

For the transitions for the rest of the games, we are just reversing the changes made in previous games and thus these previous proofs suffice to show that they will also be indistinguishable. In particular, the proof of indistinguishability of Games G_7 and G_8 (in which we switch back to encryptions of the honest values rather than random encryptions) is analogous to the proof of Lemma G.6; the proof of indistinguishability of G_8 and G_9 (in which we switch back to decrypting rather than extracting) is analogous to the proof of Lemma G.5; the proof of indistinguishability of G_9 and G_{10} (in which we switch back to re-randomizing rather than freshly encrypting) is analogous to the proof of Lemma G.3; the indistinguishability of G_{10} and G_{11} (in which we return

to using honest proofs in the regular shuffle oracle) is analogous to the proof of Lemma G.2; and the indistinguishability of G_{11} and G_{12} (in which we return to using honest proofs in the initiation oracle) is analogous to the proof of Lemma G.1.